NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

DSS DEVELOPMENT EFFORTS AT
THE MARE ISLAND NAVAL SHIPYARD

by

Michael F. Rall
and
Richard N. Woodman

March 1987

Thesis Advisor: Norman R. Lyons

Approved for public release; distribution is unlimited.

unclassified

TY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

PORT SECURITY CLASSIFICATION unclassified	1b. RESTRICTIVE MARKINGS
CURITY CLASSIFICATION AUTHORITY	3 DISTRIBUTION / AVAILABILITY OF REPORT Approved for public release; distribution is unlimited.
CLASSIFICATION / DOWNGRADING SCHEDULE	
FORMING ORGANIZATION REPORT NUMBER(S)	5 MONITORING ORGANIZATION REPORT NUMBER(S)
ME OF PERFORMING ORGANIZATION 1 Postgraduate School 6b OFFICE SYMBOL (If applicable) 54	7a NAME OF MONITORING ORGANIZATION Naval Postgraduate School
DRESS (City, State, and ZIP Code) Monterey, California 93943-5000	7b. ADDRESS (City, State, and ZIP Code) Monterey, California 93943-5000
ME OF FUNDING / SPONSORING GANIZATION 8b OFFICE SYMBOL (If applicable)	9 PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
DRESS (City, State, and ZIP Code)	10 SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO / PROJECT NO / TASK NO / WORK UNIT ACCESSION NO

LE (Include Security Classification)
DSS DEVELOPMENT EFFORTS AT THE MARE ISLAND NAVAL SHIPYARD

SONAL AUTHOR(S) Rall, Michael F. and Woodman, Richard N.

PE OF REPORT	13b TIME COVERED	14 DATE OF REPORT (Year, Month, Day)	15 PAGE COUNT
ter's Thesis	FROM _____ TO _____	1987 March	308

PLEMENTARY NOTATION

COSATI CODES			18 SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
LD	GROUP	SUB-GROUP	Decision Support System (DSS); budget development and control

TRACT (Continue on reverse if necessary and identify by block number)

mandate of a cost conscious Congress and American people caused SEASYSCOM to commission a study to identify areas for improvement within Naval Shipyards. Budget development and control was one area ntified. The focus of this thesis is centered on a single shipyard, Mare Island facility, detailing the budgeting operations of one of its artments. The objective is to develop an initial pilot project, a totype Decision Support System (DSS), that will address the concerns budget preparation, control and variance analysis. Additionally, this ject assesses the feasibility of larger DSS efforts within the shipyard. methodology of the development was a blend of structured and typical approaches, providing flexibility with rigorous documentation. Further ort toward integrating the findings of this thesis with the present ounting system is recommended to expand the use of decision support hin the shipyard.

RIBUTION / AVAILABILITY OF ABSTRACT NCLASSIFIED/UNLIMITED ☐ SAME AS RPT ☐ DTIC USERS	21 ABSTRACT SECURITY CLASSIFICATION unclassified
ME OF RESPONSIBLE INDIVIDUAL f. Norman R. Lyons	22b TELEPHONE (Include Area Code) (408) 646-2666 22c OFFICE SYMBOL Code 54Lb

RM 1473, 84 MAR 83 APR edition may be used until exhausted
All other editions are obsolete

SECURITY CLASSIFICATION OF THIS PAGE
unclassified

Approved for public release; distribution is unlimited.

DSS Development Efforts at The Mare Island Naval Shipyard

by

Michael F. Rall
Lieutenant, United States Coast Guard
B.S., United States Coast Guard Academy, 1981

and

Richard N. Woodman
Captain, United States Marine Corps
B.A., St. Lawrence University, 1977

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN INFORMATION SYSTEMS

from the

NAVAL POSTGRADUATE SCHOOL
March 1987

ABSTRACT

The mandate of a cost conscious Congress and American people caused NAVSEASYSCOM to commission a study to identify areas for improvement within US Naval Shipyards. Budget development and control was one area identified. The focus of this thesis is centered on a single shipyard, the Mare Island facility, detailing the budgeting operations of one of its departments. The objective is to develop an initial pilot project, a prototype Decision Support System (DSS), that will address the concerns of budget preparation, control and variance analysis. Additionally, this project assesses the feasibility of larger DSS efforts within the shipyard. The methodology of the development was a blend of structured and typical DSS approaches, providing flexibility with rigorous documentation. Further effort toward integrating the findings of this thesis with the present accounting system is recommended to expand the use of decision support within the shipyard.

THESIS DISCLAIMER

The reader is cautioned that computer programs developed in this research may not have been exercised for all cases of interest. While every effort has been made, within the time available, to ensure that the programs are free of computational and logic errors, they cannot be considered validated. Any application of these programs without additional verification is at the risk of the user.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

I. INTRODUCTION

A. BACKGROUND

In past years, Naval Shipyards operated in a "zero gain/zero loss" mode. That orientation is rapidly changing. The thrust of this change is that Defense Department managers must be concerned with costs. In keeping with this spirit, NAVASEASYSCOM commissioned a study of US Naval Shipyards to identify functional areas needing improvement. The contracted analysts investigated all areas of the shipyards and found numerous problems. In the MIS systems area it was found that the central system was behind technology and producing poor management reports.

Budget development and control was also found to be lacking. In the past there was little incentive to be concerned with budgets, since cost was not a critical concern. But budget preparation is particularly important in maintaining control of costs. There will be little improvement unless a budget and expense monitoring capability is provided for the shipyard managers. In light of these concerns and developments, this study undertook to develop and investigate the feasibility of a Decision Support System (DSS) that would assist managers at the shipyard with budget preparation and control. Additionally, the specific methods of cost classification at Naval Shipyards are discussed. These classifications are by Cost Center, Cost Function, and Cost Class, and are the vehicles by which costs are portrayed.

Within the Management Engineering and Information Office (MEIO), at the Mare Island Naval Shipyard, budget development and control is performed by the department budget analyst with inputs from the department managers. Departmental budgets are presently developed annually. The primary budget process inputs originate from the Comptroller with additional inputs from within MEIO, although these are not formal inputs.

Budget analysts are also responsible for providing responses to ad hoc queries from the shipyard and department managers. Those queries can come in the form of requests for cost accrual analysis to investigations of variances the managers feel may be potential problem areas.

The Shipyard Automated Budget Reporting System (SABRS) is a new addition to the shipyard's accounting systems. It will allow in depth cost analysis, and assistance in the preparation of budgets. Presently, not all of its features are operational.

B. OBJECTIVES

The primary objective of this thesis is to develop an initial pilot project, a prototype DSS, that will initially address the concerns of budget preparation, control and variance analysis within the Mare Island Naval Shipyard. In addition, it is intended that this DSS will be the first step in a larger effort to provide managers access to an easy to use, graphically oriented, and organization-wide system. With these objectives in mind, the final objective will be to assess the applicability of larger DSS efforts within the shipyard.

A byproduct of the project will be to provide understanding and insights into the use of programs that are on-hand at the shipyard and incorporated into the DSS. We feel that, in the past, most organizations have had a tendency to go for the salesman's hype, rather than exhausting the possible capabilities of present systems first. The hope is that the users will at least be given the tools with this DSS to further explore on-hand programs.

C. RESEARCH QUESTIONS

The most important question to answer in this research is, what is the best method to integrate the available tools to produce a coordinated, specific DSS? The insights gained from this effort will identify the best way to develop future DSS's.

Secondly, we attempted to investigate the most efficient methodology of system development. To do this we followed the iterative approach identified by Keen, Scott Morton [Ref. 1] and Sprague and Carlson [Ref. 2]. Also, we elected to use the "structured techniques" of Yourdon [Ref. 3] and De Marco [Ref. 4] where applicable. The structured approach is more rigorous in its requirement for documentation. This will be prove to be beneficial for the users in enhancing their understanding, and for later developers conducting maintenance and expansion.

D. SCOPE AND LIMITATIONS

We approached this project as a prototype, an experiment to determine the applicability of a DSS to shipyard-wide decision support for management. The intent

was to learn from this experiment, refine the results, and subsequently expand the system. In effect, this project was a feasibility study.

In order to meet the requirements of resources and time, the scope of this project was limited to the budget analysis and control requirements of MEIO. This fact does not diminish the information to be learned.

The DSS was implemented on two systems: a PRIME 9755 minicomputer which is linked in a shipyard-wide Local Network and a standalone IBM microcomputer configuration. The requirement to use the PRIME was a limitation because it required us to learn a new system and language. It was also an opportunity since any useful systems developed could easily be used by any other department on the Local Network. Additionally, as mentioned earlier, the minicomputer implementation made use of programs presently available at the shipyard. The resource constraints and the experimental nature of this project precluded purchasing software that would meet the needs of shipyard managers.

The portions of the project written by the authors were done in PRIME's Command Processor Language (CPL) for the minicomputer and C for the microcomputer. They allowed the use of structured programming techniques and constructs, which we felt were essential to producing code with minimal errors. Both of these helped to support the rapid development approach required with the iterative prototyping methodology.

E. LITERATURE REVIEW AND METHODOLOGY

1. Literature Review

In an attempt to clarify the term DSS, a literature review was conducted to provide a general definition based upon those suggested by the key investigators in this area of research. DSS, as opposed to MIS, attempts to work in the vague arena of unstructured problems. To highlight the differences, the "structured techniques" were compared to the usual DSS development approach. In certain ways, we noted some parallels and similarities. An important parallel is seen in the logical first step of each. The structured approach looks at problem definition, the DSS approach to the identification of a key decision. Those starting points provide the scope within which the project developments take place.

The authors we reviewed stressed the importance of organizational approaches to DSS development strategies. Integrating DSS to the organization requires careful

planning. The technology levels, the appropriate tactical option, and a coherent "action plan" for DSS must be carefully selected. The hygiene issues facing a DSS development are critically important to the success of the system. These environmental issues require close consideration by any would-be developer.

In conducting the analysis for a DSS, we highlight one proposed by Sprague and Carlson [Ref. 2]. It is an alternative pattern of analysis which is different than the usual systems analysis methods, in that it is process independent and not data driven.

2. Methodology

The development effort of this thesis was a blend of structured and DSS approaches, using each where it was most logical. This provided us with a very flexible methodology. The cornerstone was "iterative development," which was the cement that united the two disparate methodologies we incorporated.

Our goal was to have the flexibility of the DSS approach, but retain the rigorous documentation standards of the structured techniques. Therefore, we used the structured techniques within the overall framework of the DSS approach.

We also identified the need to include data base development requirements within our methodology. The data base is important to the operation of DSS systems, so those issues should not be ignored.

F. SUMMARY OF FINDINGS

The methodology we selected provided us with a flexible, but rigorous development environment. This would essentially equate to the prototyping approach mentioned by Yourdon [Ref. 3]. For DSS, the slack produced by this methodology is essential because of the vague nature of unstructured problems.

We also found that the structured techniques that are actively employed within the general DSS framework were absolutely necessary to ensure that sufficient documentation was developed with the prototype system to assist future development efforts. In fact, the techniques fit very well with the DSS approaches we identified. Both sides of the coin are needed. The blending rests clearly with the iterative nature of DSS approaches. Without it, the reconciliation between the dissimilar approaches could not be bridged.

G. ORGANIZATION OF STUDY

Chapter Two presents the background on the Mare Island Naval Shipyard and its present systems. The reader is given an appreciation of the problems facing the

shipyard managers and how our project is an attempt to deal with one area: budget preparation, control, and variance analysis.

Under the framework identified by the background, the issues concerning DSS development in the current literature are discussed in Chapter Three. This presents the current understanding of the role of Decision Support Systems, and provides a specific definition.

Refining the general issues further, Chapter Four presents the methodology we followed in our development effort. This methodology is an outgrowth of the areas of importance identified during the literature review and the background study.

Chapter Five presents the pilot projects which we developed, defining specific design issues we faced. Chapter Six develops an analysis of the projects presented in Chapter Five. We focus on lessons learned and identify key areas of concern for future developers.

II. BACKGROUND

A. THE COOPERS AND LYBRAND STUDY

During 1985, the Coopers and Lybrand accounting firm conducted a management analysis of all US Naval Shipyards. The scope of the analysis consisted of all functional areas of shipyard operations and management.

In the MIS arena, they found that the central system is woefully behind technology. It produces reports which are not timely and are marginally useful. Additionally, NAVSEASYSCOM gave little direction concerning the growth and acquisition of the hardware and software components of computer systems for the shipyards. Consequently, both external and internal incompatibility resulted.

Budget development, execution and control were also found to be lacking. In the past, managers had little incentive to keep budgets under control since cost was not a critical management concern. The present budget concerns of the Congress and the American people have resulted in the need for all government agencies and activities to take a closer look at the management of their operations, and take the necessary actions to rectify situations where resources are wasted.

As stated in the Coopers and Lybrand report [Ref. 5: p. FIN-21]:

"The (present) budget process does not support meaningful variance analysis. Several factors contribute to this problem:"

> The budget is perceived by many shipyards as a funding tool and is not effectively used as a planning and control tool.

> The budget loses credibility as an effective management tool because it does not adequately account for the impact of changes in demand. It is not capable of effectively supporting analyses of spending and volume variances, particularly at the department of responsibility center level.

Specifically associated with managing costs in the shipyards was the lack of sufficient incentives "for departments to come in under budget."

> The budget is viewed as a spending limit, not a spending target...The NIF budget policy manual [Ref. 6] even states that 'the budget is a plan of the activity to attain a cumulative no gain/no loss position at the end of the budget year.' The manual says further that 'the established overhead rates should be developed...so that zero balance variance between actual and applied overhead is achieved at year end to avoid the requirement of distribution thereof'.

This attitude encourages department managers to spend to their budget limits because they believe that favorable budget variances frequently result in a reduction in subsequent budget allowances. While the financial systems are capable of isolating poor performance, they are not effective at identifying and rewarding good performance. The manager who does have a favorable budget variance, not only does not get favorable recognition, but may also find himself with reduced funds the next fiscal period

Budgets were prepared quarterly or annually before the study was initiated. The management analysts of Coopers and Lybrand were concerned by this practice. They believed monthly budget preparation would improve the quality of the planning process [Ref. 5: p. FIN-17-18].

"Departments are not required to justify projected costs in sufficient detail. Projected costs are not based on key activity indicators." [Ref. 5: p. FIN-18] Managers need some method of analyzing various costs in order to analyze variances, to project future budgets and to defend those budget projections. They need performance measures or indicators to chart their current status.

The study also reported that managers were not generally involved in the preparation of their departments' budgets. "Since these managers lack a sense of ownership for the budget, it loses some of its effectiveness as a motivation tool." [Ref. 5: p. FIN-19]

Budget effectiveness was reduced because of the poor quality of budget submissions. Many budgets required later revision. "In addition, shipyard management emphasizes total departmental control points. Little emphasis is placed on adherence to the budget on a line item basis. For example, an analysis of the year end actual versus budget report for a shop in one shipyard identified by the study [Ref. 5: p. FIN-20]:

1. Six line items with expenses up to $107,000, but no budgeted amounts.
2. Twelve items with ranges of favorable budget variances, from $3000 to $248,000.
3. Thirteen line items with unfavorable budget variances from $1000 to $142,000.

Budget variances are determined by comparing budgeted amounts to actual expenses. If the actual expense is less than the budgeted amount, a favorable variance exists. An unfavorable variance occurs if the actual expense is greater than the budgeted amount. Unfavorable variances are to be avoided.

B. COST CLASSIFICATION

Two important means of classifying costs are by Cost Function and by Cost Class. Cost functions group costs into functional areas. Each Cost Center, which is equivalent to a department, has several Cost Functions under it, depending on the nature of its work. For example, the MEIO is Cost Center 9110. Under its control are eight Cost Functions designated numerically from 9112 to 9119. Table 1 shows the names of these Cost Functions.

TABLE 1
COST FUNCTIONS

9112	Administration
9113	Administration
9114	Rental and Maintenance of ADP Equipment
9115	Operations
9116	Control and Scheduling
9117	EDP Operations
9118	Key Entry Operations
9119	NAVSEA NSY MIS Program

Each expense incurred by MEIO falls under one of the listed Cost Functions. The sum of the expenses of all Cost Functions under a Cost Center is the amount expended by that Cost Center. Cost Functions relate to Resource Management System functional/subfunctional categories defined inthe Navy Comptroller's Manual [Ref. 7].

Cost Class is another way expenses are classified. A Cost Class is the identification of the type of an expense. They relate to elements of expense, under the RMS system, "which tell what kind of resources are used [Ref. 7]." Each Cost Center has certain authorized Cost Classes. MEIO, for example, is authorized 21 different Cost Classes under which it may spend. Table 2 lists the authorized Cost Classes for Cost Center 9110.

TABLE 2
COST CLASS

02	Supervision, Graded
03	Non Supervision, Graded
04	Non Supervision, Ungraded
10	Lost Time
11	Time Allowed
12	Consumeable Supplies
19	Coding Rejects
23	Union Activities
28	Alterations
30	Travel
32	Rent and Communications
33	Printing, Reproduction, and Duplicating
39	Training, Other
43	Depreciation of Purchased Equipment
54	Shop, General Non Labor
68	Acquisition of Minor Property
91	ADP Supervision
92	ADP Analyst/Programmer
93	ADP Operations
94	ADP Rent/Communications
95	ADP Maintenance
96	ADP Contractual Services
97	ADP Consumeable Supplies and Installation
98	ADP Minor Property
99	ADP Training

All expenses are thus classified under both Cost Function and Cost Class. Only certain Cost Classes are authorized under a given Cost Function however. A Cost

Class can be present in many Cost Centers, unlike Cost Functions which are related to their particular Cost Center only. Table 3 shows the authorized Cost Classes for the Cost Functions under the Cost Center 9110.

TABLE 3

AUTHORIZED COST FUNCTIONS/COST CLASSES
Y = AUTHORIZED N = NOT AUTHORIZED

Cost Class	Cost Function 9112	9113	9114	9115	9116	9117	9118	9119
02	Y	Y	N	Y	Y	Y	Y	Y
03	Y	Y	N	Y	Y	Y	Y	Y
04	Y	Y	Y	Y	Y	Y	Y	Y
11	Y	Y	N	Y	Y	Y	Y	Y
12	Y	Y	Y	Y	Y	Y	Y	Y
23	N	Y	N	N	N	Y	Y	N
28	Y	Y	Y	Y	Y	Y	Y	Y
30	Y	Y	Y	Y	Y	Y	Y	Y
32	N	Y	Y	Y	Y	Y	Y	Y
33	Y	N	Y	Y	Y	Y	Y	Y
39	Y	Y	N	Y	Y	Y	Y	Y
43	Y	Y	Y	Y	Y	Y	Y	Y
54	Y	Y	Y	Y	Y	Y	Y	Y
68	Y	Y	Y	Y	Y	Y	Y	Y
91	Y	Y	N	Y	Y	Y	Y	N
92	N	Y	N	N	N	N	N	N
93	Y	Y	N	Y	Y	Y	Y	N
94	N	N	Y	N	N	N	N	N
95	N	N	Y	N	N	Y	N	N
96	Y	Y	N	N	N	Y	N	N
97	Y	N	N	N	N	Y	N	N
98	Y	N	Y	N	N	N	N	N
99	Y	Y	N	N	N	Y	N	N

C. THE PRESENT MEIO BUDGET CONTROL AND DEVELOPMENT PROCESS

As pointed out by Coopers and Lybrand, the department manager is responsible for his department's budget, and therefore should be involved in the budget development process. His involvement comes in the form of monitoring the development process.

The budget control and development activity within MEIO involves two key groups: the departmental managers and budget analyst. In most cases their give and

take in identifying resources required develops the budget over time. Various managers submit their sections' needs and the budget analyst reconciles those with direction from top department management and the Controller department. The key to this process is "that the lines of budget submission and approval must follow the lines of organizational responsibility." [Ref. 7: p. C-4] This process is graphically depicted in the structured specification in Appendix A.

Departmental budgets are prepared annually and submitted to the shipyard Comptroller. They are developed by the particular department's budget analyst with various inputs and constraints presented from within the department and without. Budget request inputs and clarifications are received from departmental managers. Specific MEIO requirements and constraints are received from the top departmental management. The Comptroller's office submits various budget constraints to all shipyard departments. These include dollar ceilings and floors, and leave hours not to exceed 14% of the total budget for labor.

Initially, when we were studying the process to develop the descriptive analysis, budget input reports were produced and distributed by the Data Processing section of MEIO. These reports were developed from information submitted by the Comptroller's office. With the advent of a new shipyard accounting system, SABRS, the majority of these reports were consolidated into one, and now originate directly from the Comptroller department. At the present time the report is printed and distributed manually; however, in the future it is anticipated that it will be electronically distributed through the Prime network.

At the outset of our analysis, SABRS was still in the coding and testing stage but past the projected implementation stage. The original contractor had been replaced and the learning curve for the new programmers delayed the implementation. SABRS was implemented at the beginning of fiscal year 1987, which was in the middle of our system development. Consequently, SABRS is the information generation system that we shall consider throughout this discussion, and not the old reports system.

After the budget input reports are received, they are used to track budget performance. Performance reports compare budgeted to actual expense by Cost Function, Cost Class, Cost Function/Cost Class, Cost Center, or by total for the shipyard. These reports also serve as the basis for developing reports in response to departmental managers' queries. These queries are ad hoc in the sense that they are not formalized, but may be requested when managers feel a need to monitor a

particular aspect of the budget. For example, prior to the implementation of the SABRS system, the shipyard Commanding Officer requested a breakdown of costs for each department by Cost Classes. The reports used at that time did not have the information summarized by Cost Class for each department and the computer center did not have the time to handle this one time request. Therefore, the information had to be extracted by hand from the existing reports. This was not only time consuming for the budget analysts, but also demonstrates the state of the current computer system. The desired data was on the computer but there was no easy way to extract it electronically.

Potential problems are detected by either the managers of a particular department or by the budget analyst. If a manager finds a potential problem in the performance report, he will contact the budget analyst who will research the problem. The reason causing an unfavorable variance can be determined, and the manager can decide if action is warranted to correct the problem. Or the "problem" may be the result of planned expenses and not a real problem at all.

Likewise, if the budget analyst detects a potential problem, she will alert the appropriate managers. When further information is desired for clarification, job order information that is associated with the expense may be accessed.[1] By accessing on-line job order information, detailed information on an expense can be obtained. This ideally will clarify any questionable variances.

D. SABRS

SABRS is a newly implemented accounting system with some interactive capability. It is primarily intended to be used by the shipyards in the preparation of their annual budgets for submission to NAVCOMPT. It is an on-line system with several capabilities.

SABRS I is to be utilized by the budget officers and department budget analysts to project costs using prior year's actual costs, escalation, and acceleration projection as well as anticipated workload. The budget can be created from scratch, from existing information, or from previous budgets. The historical data base is invaluable in analyzing trends and making budget projections based on those trends. This historical data base will be limited to only the previous year.

---

[1]All work done at the shipyard is assigned a job order number. A job order number consists of the Cost Function number, Cost Class number, and a four digit number.

This interactive system will allow the user to test budget proposals and conduct "what if" analysis of various budget options. Budget detail can run the spectrum from shipyard total budget, to Cost Center, Cost Function and Cost Class levels. Graphic display of the resulting analysis is not available with the system.

Although the system has many interactive capabilities for the various users, at present it is only used to create budget versus actual performance reports (SBR-22A and SBR-22B). Obviously, the complete system implementation of SABRS I will be required before the shipyard managers have a comprehensive and sophisticated budget analysis tool.

SABRS II is designed to allow the users to prepare budget exhibits required by NAVSEASYSCOM. Although it will provide assistance to the Comptroller's office and department budget analysts, its analysis capabilities for the average user are limited.

III. LITERATURE REVIEW AND THEORETICAL FRAMEWORK

A. DSS DEFINITION

Many books and articles have been written on DSS. The subject is relatively new, becoming part of the Information Systems jargon as recently as the mid 70's. The exact definition of what can be classified as a DSS is itself a gray area.

Peter G. W. Keen and Michael S. Scott Morton [Ref. 1] are acknowledged as the major authorities on DSS. Their work was one of the first major writings on the subject of DSS and is used as a focal point for most subsequent works. They originated the now standard term DSS. Keen and Morton [Ref. 1: p. 15] view DSS as part of the natural evolution and maturation of information systems and management. Mature MIS systems should be available before a DSS system can be implemented with success.

A DSS has been characterized as an interactive, computer-based system that helps decision makers utilize data and models to solve unstructured problems [Ref. 2: p. 4]. A system requires three capabilities to be classified as a DSS: a data base management system, a model base management system, and a dialog management system [Ref. 2: p. 28]. MIS deals with structured tasks, while a DSS deals with semi-structured or unstructured tasks. This divergence from what is considered MIS has caused the development of new techniques and methodologies for the development of DSS. These techniques often are not new to the MIS community, but rather have been considered improper for the development of past MIS.

B. DESIGN AND IMPLEMENTATION

1. Structured Approach

Structured analysis and design is the present popular and successful procedure for the development of MIS. Structured analysis and design provide a checklist for the developer to insure all necessary aspects of the system are incorporated. Although often given other names, the following steps are generally associated with the structured approach [Ref. 8: p. 17]:

1. Problem definition
2. Feasibility study
3. Analysis
4. System design

5. Detailed design
6. Implementation
7. Maintenance

Since DSS is concerned with problems that are classified as semi-structured or unstructured, it seems logical that a structured approach may not be the best approach. A structured approach requires that each step be methodically followed as they are listed. The output for one step serves as the input for the next step in this top-down approach. Often it is not easy, if not impossible, for an unstructured or semi-structured task to be fully defined. Therefore, it can be very difficult to develop a clear design flowing from step to step. The nature of unstructured and semi-structured task definition involves backtracking through the development steps, following an order which is not consistent with a top down approach. A structured approach may indeed have frequent returns to earlier steps, but the sense of progression from step to step must be maintained [Ref. 8: p. 16]. This progression does not have to be maintained in DSS development.

2. DSS Approach

Keen and Morton's process for developing a DSS is not a cook book methodology. There is not as clear cut a process for the design of a DSS, although several authors note common, key processes that must be accomplished sometime within the development cycle.

The main distinction between the current trends and practices in the MIS field and in DSS design can be seen in the concurrency of design and implementation. Design and implementation are inseparable and evolutionary in DSS [Ref. 1: p. 167]. This is a common thread in the DSS literature. Design and implementation are not separate phases but two blended iterative steps.

The unstructured nature of DSS problems can result in vague problem definitions. This vagueness can only be approached through the flexibility of this iterative DSS methodology. The concurrency of design and implementation act as a bridge between the designer and the user, increasing their communication. Short as possible cycles of design and implementation allow the users to frequently evaluate the development effort and increases understanding of the users' needs.

Two other important authors of DSS literature who built on the ideas proposed by Keene and Scott Morton are Ralph H. Sprague, Jr. and Eric D. Carlson. Sprague and Carlson [Ref. 2], also emphasized the need for an iterative design. A DSS

must "be built with short, rapid feedback from users to ensure that development is proceeding correctly" [Ref. 2: p. 15]. This keeps the user involved and aware of the progress, making change easier and quicker. The stages of typical system development, analysis, design, construction, and implementation must be "combined into a single step which is iteratively repeated" [Ref. 2: p.15]. This is similar to the concept expounded by Keene and Scott Morton, except expanded to be more inclusive. Sprague and Carlson not only included design and implementation in their iterative methodology, but also analysis.

3. **Problem Definition**

Focusing the Approach structured approach commences with the definition of a problem. Although Keen and Scott Morton's approach is dissimilar to a structured approach, it does have a logical first step: the identification of a key decision [Ref. 1: p. 173]. A key decision, once identified, becomes the focus of the initial DSS. This narrows the scope of the initial implementation and allows the user the flexibility to make changes after he sees a working system. Any changes can be designed and implemented in an iterative fashion.

When choosing what decision to implement, the probability of success is always increased if the client has a readily identifiable problem or need. Normally this results in a user who is an excited proponent of the new system. The "anything is better than what I have now" attitude often breeds cooperative clients and sometimes even fanatics. A cooperative and committed user does not guarantee a successful project, but without some support from the user, the project is doomed to fail. The earlier his commitment is generated, the easier the development will be.

4. **Strategic DSS Development**

Sprague and Carlson expanded on the concept of focusing on one key decision and divided DSS into three technology levels: specific DSS, DSS generators, and DSS tools. The level that actually accomplishes work is the specific DSS level. This is the final product that serves the user's needs. Specific DSS's are built from other levels, either from generators or tools. The lowest DSS level is that of tools. Tools facilitate the development of either Specific DSS or a DSS generator. A DSS generator is a collection of tools or capabilities. [Ref. 2: pp. 10-12]

When developing a DSS several approaches can be taken. First, tools can be developed with no specific DSS in mind. These tools can then be integrated to build a DSS generator. The generator can then be applied to several different specific DSS's

as applications arise and are implemented. New tools can be added to give the generator more power. This approach is expensive and tangible results are slow in developing. Careful planning and extensive analysis is necessary to insure the proper tools are available when a DSS application arises.

The other approach focuses on a specific DSS, acquiring the tools needed to build that specific DSS's. These tools can then possibly be used for other specific DSS's within the organization. A generator is then created indirectly by combining tools. A generator does not evolve until several specific DSS's are developed. Building a specific DSS is the preferred method because immediate results can be seen quickly and at a relatively inexpensive price. Subsequent specific DSS development is not as simple as it would be if a generator already existed.

5. **Tactical DSS Development**

Depending on environmental factors (such as the organizational structure, the tasks, the users, and the builders) three different tactical options are recommended by Sprague and Carlson [Ref. 2: p. 60]:

1. The quick hit. If it is not clear that a general DSS capability is needed, but there is a recognized highpayoff area for decision support, develop a Specific DSS directly using the most appropriate tools, capture the benefits, then consider what to do next.
2. The staged development approach. Build one specific DSS, but with some advanced planning, so that part of the effort in developing the first system can be reused in developing the second. With appropriate planning, a DSS generator evolves from the development of several successive specific DSS.
3. The complete DSS. Before building any specific DSS, develop a full-service DSS generator, and the organizational structure for managing it.

The staged development is recommended because it is the most balanced approach of the three. Results can be seen quickly and future DSS development is planned, which is usually less expensive than building a system without any planning.

6. **A DSS Action Plan**

Planning is the key to success in any endeavor, and design of DSS is no exception. Sprague and Carlson [Ref. 2], have identified four phases of an action plan. These phases do not conflict with the iterative, simultaneous design and implementation of DSS. Rather, they provide a framework for the design of additional DSS. The four phases of the plan are [Ref. 2: pp. 67-68]:

1. Preliminary study and feasibility assessment.
2. Development of the DSS environment.
3. Developing the initial specific DSS.
4. Developing subsequent specific DSS.

The preliminary study and feasibility assessment are the same as in the structured methodology. Once a problem is identified, a feasibility study is needed. In fact, this study can be initiated prior to the knowledge that a DSS is necessary or desired, in response to a particular problem. During this preliminary study, pilot projects can be implemented to ascertain DSS needs. These pilot projects can also help find the project for the first specific DSS.

Phase 2 forms the DSS environment. A minimal set of tools are either developed or purchased with a plan for creating a DSS generator. The initial specific DSS design and implementation can then begin. The specific DSS should be in a highly visible area that has observable benefits. Tools can be updated or added as necessary.

In the next phases the specific DSS is designed and implemented in an iterative methodology. Upon completion of the initial specific DSS, other DSS's can be developed. The second DSS should be related to the first. For example, duplicating the same system for another division with a different group of users would probably be easier and yield a quicker payoff than developing an unrelated system [Ref. 2: p. 67]. In practice the initial DSS does not have to be completed prior to the start of phase 4.

7. **ROMC**

Another technique Sprague and Carlson introduced was the ROMC process [Ref. 2: p. 96]. ROMC is a process independent approach for identifying the necessary capabilities of a specific DSS by focusing on Representations, Operations, Memory aids, and Controls. The representations help to conceptualize and communicate the problem or decision situation. They identify what the user actually wants to see as output to make better decisions. Operations allow the user to analyze and manipulate those representations. They allow the user to integrate the representations desired into the decision maker's style. Memory aids assist the user in linking the representations and operations together. A data base acts as a memory aid by presenting different views, profiles, libraries, or by acting as a trigger that reminds the user that certain operations need to be performed. Control mechanisms handle and integrate the entire system. Control mechanisms allow the decision maker the use of the system in a way that conforms to a particular user's decision style. Menus, training, and some operations make up the control mechanisms for a DSS.

C. AREAS OF DESIGN

The design must also cover three almost separate areas [Ref. 1: p. 181]:

1. the "user" design, defined in terms of the imperatives,
2. the interface or driver which links them, and
3. the data-base management design.

Each of these areas must be addressed in the design phase.

If user design, the data base, or the interface is ignored, the consequences could cripple further DSS development. When the user's needs are not fully addressed, he will not use the system. Neglecting data base management results in the necessary information not being available when needed. If the driver system which links all of the modules for the entire DSS is not designed properly, then the system cannot function as a coherent system, even if all its parts work perfectly.

1. User Design

The user design is the logical design of the system. This develops the tools that form the DSS. The developers must have a clear understanding of what the user's needs are, and ensure that these are addressed in the system design. That requirement is no different for DSS development than for MIS development. The difference is that it is critically important for DSS. Unlike an MIS project, you can not make do with a system that meets only part of the user's requirements. It either meets those needs and is used, or it is not used.

2. System Interface

A driver links the tools into a usable system. Almost all known DSS's use some version of command-driven approach for user interaction [Ref. 1: p. 181]. This allows flexibility and ease of use. It also nurtures the learning process about the types of decisions that can be addressed.

Flexibility is necessary in a DSS because of the unstructured tasks it is designed to perform. The ad hoc capabilities, that have been the hallmark of DSS over the past decade, can only be accomplished through the use of command driven systems. Systems which are limited only to menus, do not have this flexibility. The designer cannot think of every possible combination that a decision maker may want at some future time. Even if all possible combinations are known, the cost of implementing them in another form other than through a command language would be prohibitive.

3. Data Base Management

Data-base management is of prime importance. Almost all DSS's are centered around a data-base system. The data base contains the information that the decision maker needs in order to make better decisions. The data base constitutes the capabilities of the entire system. The information that must be in the data base needs to be identified early in the design process.

Having a data base prior to the building of a DSS simplifies the design of the DSS. The functions that the DSS has to deal with are lessened. Most data base problems can then be discounted or at least simplified. The collection and maintenance of data used by the DSS may already be accomplished. Most data base management issues will be resolved, such as integrity and security. An added benefit is that the chances of different specific DSS's sharing data increases. [Ref. 2: p. 223]

Data can be obtained from many sources. A DSS must be able to aggregate a variety of information. Data can be extracted from a data base whether in house or from outside sources. The DSS is then able to manipulate data from several sources, and format it into useful information for the decision maker, without affecting the source data base. Data extraction, taking data from one data base for use in another, allows source data files to be organized for efficient data entry, update, processing, output, or protection, without additional indices or data to support the DSS [Ref. 2: p. 248].

The original data base management system can be used for the functions for which it is best suited. Indices and data for the DSS are stored separately from the original data base, within an extracted data base. This prevents degrading the performance of the original data base.

D. ENVIRONMENT

1. Attitudinal Direction and Requirements

The initial implementation using the iterative development approach is most effective in a computer resource environment that is both centralized and decentralized, and having slack resources for research projects. [Ref. 1: p. 236]. The centralized aspect of the computer resource must be at the lower tier, providing controls over the system through well structured procedures. The decentralized aspect of the computer resources provides an environment that is more attuned to experimenting, and research and development. Slack is also necessary because the DSS changes the organizational

emphasis from efficiency to effectiveness. An increase in effectiveness is usually attained at the expense of efficiency. Extracted databases, modeling, and other computer manipulations and displays can greatly decrease the efficiency of an MIS. This new DSS application should be classified as R&D until it becomes an integrated system. The slack resources cushion the inefficiencies of a DSS development preventing a drain on resources from other areas.

2. **Developmental Requirements**

Iterative design is the key to DSS design. The evolutive approach is a methodology based on the progressive design of a DSS, going through multiple, as short as possible cycles. Successive versions of the system under construction are then utilized by the end-user. The steps in the process include [Ref. 2: p. 139-140]:

1. Identify an important subproblem.
2. Develop a small but usable system to assist the decision maker.
3. Refine, expand, and modify the system in cycles.
4. Evaluate the system constantly.

These steps reiterate the previously mentioned design criteria expounded by both Keene and Morton, and Sprague and Carlson.

Selecting a problem with a high probability of success is paramount in DSS design. When a small system works and the contributions of a working DSS system can be demonstrated and observed first hand, the product sells itself.

3. **DSS Prototyping**

"The most successful installation technique is prototyping" [Ref. 2: p. 155]. Prototyping provides lower risks in the development by controlling excessive expectations. By actually demonstrating the feasibility, the user can see the actual benefits a system may have. With a created working system, the developer has concrete facts on which to base further DSS feasibility.

4. **User Education**

User involvement in any system development is important, and the iterative nature of DSS makes it doubly so. To insure the user stays involved and that future users can get involved, proper procedures for training must be implemented. User education on a DSS can be in three forms [Ref. 2: p. 153]:

1. A one on one or tutorial technique.
2. User courses, lectures, or professional development seminars.
3. Resident expert assistance when needed.

This education must begin at the early stages of development. This will ensure the user can benefit from the system.

A one on one tutorial is the most common and expensive training technique [Ref. 2: p. 154]. The use of training courses and seminars is another common method. An expert, either internal or external, can instruct users at all levels of the development. The expert, acting as a consultant, is a passive version of the tutorial technique. Different combinations of these techniques can be developed. The main lesson is: user education must be included in the design. This insures that those who can benefit from the use of the DSS have the ability and skills necessary to do so.

5. **Support for DSS**

Factors which increase the likelihood for success of a project are top management support, a clear felt need by the client, an immediate, visible problem to work on, an early commitment by the user, a conscious staff involvement, and a well institutionalized OR/MS or MIS group [Ref. 1: p. 196]. Support from top management greatly enhances the acceptability of any new project. Employees are more willing to support the development effort if they think it is a good idea. If top management is not supportive, or is hostile, others will be less likely to accept or even give the project a fair trial.

E. FINAL WORDS

Gad Ariav and Michael J. Ginzberg [Ref. 9], further define the three functions of a DSS first expounded by Keene and Morton. Figure 3.1 lists the components of a system which Ariav and Ginzberg think are necessary in order to have a DSS.

The first component, dialog management, is broken down into user interface, dialog-control and request transformer. Without each of these subcomponents the dialog management system is incomplete. Data management, the second component, is subdivided into the data base and data base management system, a data directory, a query facility, and a staging and extraction function. Finally, the model management system is partitioned into a model-base management system, model execution, modeling command processor, and a data base interface. These subcomponents define the complete DSS.

Ariav and Ginzberg [Ref. 9], looked at the basic resources of a computer system: hardware, software, people, and data. They found it is only after the DSS has been designed (after the components and their "ideal" arrangement have been selected) that

1. Dialog Management
    1. user interface
    2. dialog-control
    3. request transformer
2. Data Management
    1. data base and dbms
    2. data directory
    3. query facility
    4. staging and extraction function
3. Model Management
    1. mode-base management system
    2. model execution
    3. modeling command processor
    4. data base interface

Figure 3.1 DSS Components (Functions).

resources should be considered. [Ref. 9: p. 1049]. The solution should drive the selection of a system, the system should not drive the solution. If the arrangement is set prior to design, the options of the design are greatly hampered and creativity stymied. The designer must design a DSS based on the solution to a problem.

Ariav and Ginzberg [Ref. 9], divided software into four types:

1. General-purpose programming languages.
2. DSS tools.
3. DSS generators.
4. Generalized DSS.

These types correspond to the technology levels of DSS from Carlson and Sprague. General purpose programming languages provide only limited leverage for the development of a DSS. Tools are already available and are usually cheaper than reinventing the wheel. DSS tools provide only a single function, but as a group can be the building blocks for a DSS generator. Tools only address one function of DSS and

need to be integrated into a system. DSS generators are a collection of DSS tools but are tailored to one specific problem. For example, the DSS generator IFPS deals only with financial problems. Generalized DSS can support a class of problems. An example of generalized DSS would be a PERT/CPM system.

The design of the DSS should be independent of the software as well as the other components. This is known as an outside approach. An outside approach provides that the selection of components and their arrangement (the inside), must follow from an understanding of the environment and the role (the outside) [Ref. 9: p. 1051].

The last author and noted authority on DSS that should be looked at is George P. Huber [Ref. 10]. Huber [Ref. 10: p. 250], states that a DSS is a system specifically developed to carry out some of the decision making information processing. This definition, although vague, shows the trend for DSS's toward an increasing direct decision making influence. A DSS allows decision makers to make use of the data that other technologies are making more and more available. There is indeed an information boom as we move into a service oriented society from an industrial society [Ref. 10: pp. 250-252]. This trend, as well as the advances in technology, has provided the decision maker with more and more information. The role of the DSS is the aggregation and summarization of all the pertinent information that is available from all sources. A DSS allows decision makers to make use of the data that other technologies do not.

IV. METHODOLOGY

A. INTRODUCTION

The methodology we followed was a task organized approach. We chose our methodology as we proceeded based on our particular situation and constraints. This approach to DSS's was a semi-structured one, which is fitting since DSS is supposed to help solve semi-structured problems. The project had a purely R&D attitude, with a useful product as a possible side benefit.

Throughout this thesis we refer to our system as a prototype. A more precise terminology may be pilot project. Our intent is not to initiate a DSS but to "test the waters" to see if an environment suitable for a DSS exists. Our intent is to:

1. Test a methodology based on a semistructured, task organized approach.
2. Determine if a suitable environment for further DSS development is present or can be established.
3. Determine the next step for future DSS development, if appropriate.
4. Provide the shipyard with a usable, user friendly system involving control and graphic displays.

Since our approach in determining the environment revolved around a methodology for the development of a DSS, we refer to this system as a prototype.

Many constraints, from hardware to software, were introduced. The shipyard already had a mainframe computer, several minicomputers and microcomputers, with more micros on order. That level of prior hardware investment and the nature of our project required our system to operate on the existing hardware. The minicomputer application was chosen because of the number of terminals connected through the shipyard network and available to all managers.

Additional constraints involved software. No new software to support the prototype was planned. Therefore, the first order of business was to learn the existing system's capabilities. The software we originally focused on included CPL, the Command Processor Language for the Prime minicomputer, the Supercomp spreadsheet program, and TEL-A-GRAF, a business graphics package that was recently purchased by the Shipyard.

B. ITERATIVE DEVELOPMENT

The major theme throughout the literature on DSS development is the use of an iterative approach, which was central to our methodology. It provided flexibility because it allowed us to combine the design and implementation phases through the building of a prototype system. Since this development strategy results in a rapidly produced prototype, it should not be construed to represent a complete production model DSS. This effort merely introduces the concept of DSS within an organization with no prior development experience in this area.

The iterative approach calls for rapid succession of the system development steps, with redefinition of each step on subsequent iterations. Our only strict exit criteria was the prototype. We set no limits on the number of iterations, nor did we expect to have a completely functional DSS at the end. Our expectation was to have a useful prototype that could demonstrate one small contribution a DSS could make, and to give direction to further DSS development within the shipyard.

1. Problem Definition

The first step in any new system is problem identification and definition. The Coopers and Lybrand study outlined several problems common to most Naval shipyards. The focus of our project centered on one set of issues: the budget system for the shipyard. This project was focused for one division, but is easily expandable shipyard-wide as the prototype matures.

In addition to the Coopers and Lybrand study, the Mare Island Naval Shipyard Information Resource Management Plan [Ref. 11], outlined the need for a shipyard-wide DSS. This perceived need for a DSS and the desirability of the budget system being a part of that DSS, placed a two-fold goal on this project: provide a system to improve managerial control of the budget and expense, and to introduce the concepts and capabilities of a DSS to the shipyard.

Keen and Scott Morton refer to a "diagnostic perspective." Designers need to be sure that "they understand the realities of the decision situation...they need a descriptive model as the basis for identifying a normative direction." [Ref. 1: p. 77] Although our system is smaller than the DSS that the authors were describing a strategy for, aspects of the areas they felt must be addressed will be identified and explained. The most applicable areas are the "organizational procedures view, the political view and the individual differences perspective." [Ref. 1: p. 63]

The organizational procedures view is articulated through the use of the structured techniques of analysis. The structured specification outlines the flow of data, the processes or procedures that manipulate the data, and the storage of data. The data dictionary seeks to define the data elements identified during the data flow analysis. The structure chart shows the development of the system design to meet the requirements of the organization's operation. This effort is the structural approach practiced in present systems analysis.

The political view is approached less formally in systems analysis. It is not explicitly documented in our analysis, but has tempered our considerations and to a degree, the direction of the development effort. For example, working on a project within the Comptroller Department was not possible because of their current concerns with the SABRS project and our inability to identify definite gains or improvements for SABRS, as an outgrowth of our project. On the other hand, the degree of support available within the MEIO Department caused us to devote our initial efforts to them.

The individual differences perspective has some applicability to this project. Since our prototype was developed with inputs coming from only two individuals, we easily reconciled any particular differences between them. A larger development effort within the shipyard would require additional efforts to meet the needs of a wider spectrum of managers' styles.

2. Structured Techniques

We required that the system we developed be as maintainable as possible. The adherents to the structured methodology claim that fact; therefore, we were convinced to include the method where possible in our development. Many proponents of DSS might disagree with this decision. Sprague and Carlson point out that "DSS are,...,research efforts, not DP projects, and therefore not well suited for traditional project management procedures." [Ref. 2: p. 138] We would agree up to a point. DSS are transactionally oriented processing systems and therefore the discipline imposed by the structured techniques are still applicable. In addition, we did not have the luxury of being in close proximity to the users or the programming environment. These constraints required that we use a methodology that produced clear and understandable documentation, and also required minimal correction of the code.

We took the structured approach to analysis and design, relaxing the ending and exit criteria. The structured methodology was used for the analysis because we felt the approach of Keen and Scott Morton was not rigorous enough in identification of a

clear approach to the description process. It would not produce the documentation which we believe is essential to the clear understanding of readers and follow-on implementors. Additionally, the documentation produced would be textual. A textual description would not have been as clear, and definitely not as concise, as the structured specification.

a. Feasibility Study

If we had been closely following the structured analysis and design approach, the next step after problem definition would have been the feasibility study. Since a prototype system is an example of a technical feasibility approach, and no further systems purchases were considered, this step was not needed.

b. Analysis and Design

The next step was analysis. Our analysis took us in two directions. The budget system within the MEIO department was one area that was investigated. The other area dealt with the hardware and software that had already been selected. We were not familiar with the hardware nor the software. Hence we had to learn both the budget system and the computer system simultaneously. At this same time we began the system design. We used structured techniques to graphically map out the system. These preliminary designs were actually completed before the analysis.

The descriptive analysis we conducted corresponds to the Predesign Cycle suggested by Keen and Scott Morton. We attempted to build "momentum for change and developing a 'contract' for action..." in order to foster the initial climate for change [Ref. 1: p. 173]. The output of this approach is to produce a normative model for the present system and design the DSS in response to that [Ref. 1: p. 174]. The "degree of change" proposed by our design is not large because of the nature of our project. This approach was necessary to minimize risks and resource requirements.

To assist in this design, we followed the ROMC method introduced by Sprague and Carlson [Ref. 2]. We started at R, representations, and developed three main graphical representations. Several graphic displays of these representations were identified. We developed these graphics by focusing on one manager, a decision maker in the MEIO department. This helped us to narrow the analysis to a workable scope. Other representations were also considered, such as comparisons between budget and expenses in column form. We decided that most of that information already existed on their present reports, although not in an on line mode. SABRS was to give the user this capability when operational. We decided to concentrate on areas that were as yet

not directly addressed by the current MIS efforts. One other representation considered was the spreadsheet. A spreadsheet could give the user limited "what if" capabilities. We decided that the initial prototype would contain only the graphics in an easy to use, menu driven system.

The next step in ROMC is the Operations. For our prototype system the operations centered around the graphic display of the data. It included selecting and displaying desired budget and expense information. The Memory aids for our initial system focused on the menus of the system. Finally, the Controls for the system are also handled within the menus at this preliminary stage.

At this point in the development, the analysis of what we considered a solvable problem was completed. This problem was also considered to be of value to the shipyard. The budget problem made the expansion of the system to other departments and to the Commanding Officer of the shipyard a likely second specific DSS to pursue.

The preliminary design had been completed. The structured specification, consisting of data flow diagrams, data dictionary and structure charts had been completed. The detailed design was already begun. Basically, the simplicity of this initial system allowed us to combine these two structured steps and do them simultaneously with the analysis.

c. Implementation

The coding and implementation was then begun. This was done iteratively with revisions coming mostly through the designers. Initial graphics were completed on an IBM mainframe at the Naval Postgraduate School and then reentered on the Prime minicomputer when on site. This was done solely for the convenience of the authors. The geographic separation of the implementation site and the developers prevented a more interactive approach. This did have an effect on the timing of iterations, which caused the project to take considerably more time than initially planned. A helpful capability we had at our disposal was that we could interface directly with the shipyard's Prime minicomputer, and we authored most of the driver programs from out remote site. Some of the graphics programs were also entered from the remote site, but they could not be completely tested. The terminal used to interface with the Prime computer was a Convergent Technology's C-3 terminal and software, which was not suitable for the graphic displays produced by the TEL-A-GRAF programs.

When a skeleton working system was completed, we began a similar implementation on a micro. The micro system was developed for an IBM compatible XT or AT. Four reasons justified this effort. First, the trend toward personal computers in the work space is prevalent in all industries. Second, the shipyard just purchased several IBM compatible Zenith PC's. Third, the same analysis and design were used to code and implement the system on the PC. Fourth was our desire to quickly implement our data base design as a model of what could be developed on the minicomputer system.

These efforts confirmed the generic design of the system, which was one of our goals. We strove to design a system that could be generically implemented on any system having the appropriate capabilities.

When coding and implementation were completed the testing of the system was accomplished. Additional graphics were added to complete the final prototype. From this prototype, the direction of the future DSS development can be determined.

3. **Data Base Development**

The data base is the central focus of our proposed DSS. Further, an understanding of the data flows in the budgeting process helped us as analysts to know exactly what data were important to this system. Sprague and Carlson identified data base management as "an important prerequisite to a DSS..." [Ref. 2: p. 222]. Although we have not incorporated data extraction capabilities into our design, we would agree that a clear understanding of the data involved, and a method of manipulation of that data, is important to the design of any system.

This aspect of the system did not concern us directly. However, we did identify the data that would be needed for the initial DSS. The source of this information was also identified: the reports that the shipyard was already producing. The MIS department agreed to handle the data extraction, and we proceeded on the assumption that the data would be available in the format we desired.

The data base design efforts for the microcomputer prototype involved the relational data base methodology. Our primary reason for selecting it was our familiarity with it. In addition, it allows a great deal of flexibility to shift between logical and physical design. The relational design can also be readily applied to other methodologies. The fact that the particular relational data base management system available to us, for the microcomputer implementation, was also being installed on the shipyard's Prime network, further influenced our decision.

For our initial system the required data consisted of only expense data and budget data, which were already available. However, no historical record was kept electronically as the data tapes were overwritten about every two months. It was realized that an historical data base would need to be kept for a flexible DSS. This requirement made the choice of the minicomputer for the system even better due to the volume of the data necessary for such an historical data base. We also realized that a shipyard-wide DSS would need a flexible data base management system. Although the initial implementation would only need a simple DBMS, the backbone of an overall DSS has to be the data base system.

In the future we foresee the mainframe and the minicomputer becoming more of a data repository and less application oriented, as the microcomputers become more pervasive within the shipyard.[2] This trend would allow the decision maker to manipulate his own data base on his PC, and allow him to update or reinitialize his data base from the master file on the minicomputer.

In actual use, this data base must at least be compatible with the data base on the minicomputer. We further recommend that the two data bases have similar command languages, to avoid a situation where managers and users, have to learn two different systems. Compatible data bases would simplify the sharing of data and data extraction.

[2]Interview with Ronald Munden, Manger MEIO, Mare Island Naval shipyard, December 10, 1986.

V. PRESENTATION OF THE PROTOTYPE RESULTS

A. REQUIREMENTS DEFINITION

1. Cost Center Analysis (Minicomputer)

Cost Center Analysis (minicomputer) was developed for the Prime 9755 minicomputer. This system operates under the PRIMOS operating system and employs the Prime Command Processor Language (CPL) as its development language.

The prototype is made up of several small programs, based upon the system structure chart, in Appendix A, developed during the analysis portion of the project. Sub-programs rather than sub-routines were used because CPL does not allow the passing of parameters from sub-routines to calling program.

Differences between the modules of the structure chart and the actual programs written were the result of three factors. First, the goal when we developed the structure chart was to reduce the modules to their functional primitives. CPL has several constructs which combine some of the functional primitives. Therefore, it would have been inefficient to strictly follow the structure chart during actual coding, developing each module as a separate entity.

Second, the interface to the spreadsheet was determined to be time infeasible due to the nature of the spreadsheet's method of control. This was especially true in light of the fact that this was not a critical implementation to the users. The spreadsheet requirements could more effectively be met using SABRS. Developing the interfaces to SABRS would have been outside the scope of this initial project; therefore, it was not included.

Third, the project was originally designed to allow the user to develop his own data and include files for TEL-A-GRAF while still in CCA. The resulting loss of control, an inability to ensure the user developed standardized files, caused us to eliminate that capability. However, the flexibility required for ad hoc queries is still available within the system. This feature is provided through the TEL-A-GRAF command language.

a. Module descriptions and functions

The structure chart depicted in Figures 5.1, 5.2 and 5.3 graphically describes the interrelationships between the following program modules.

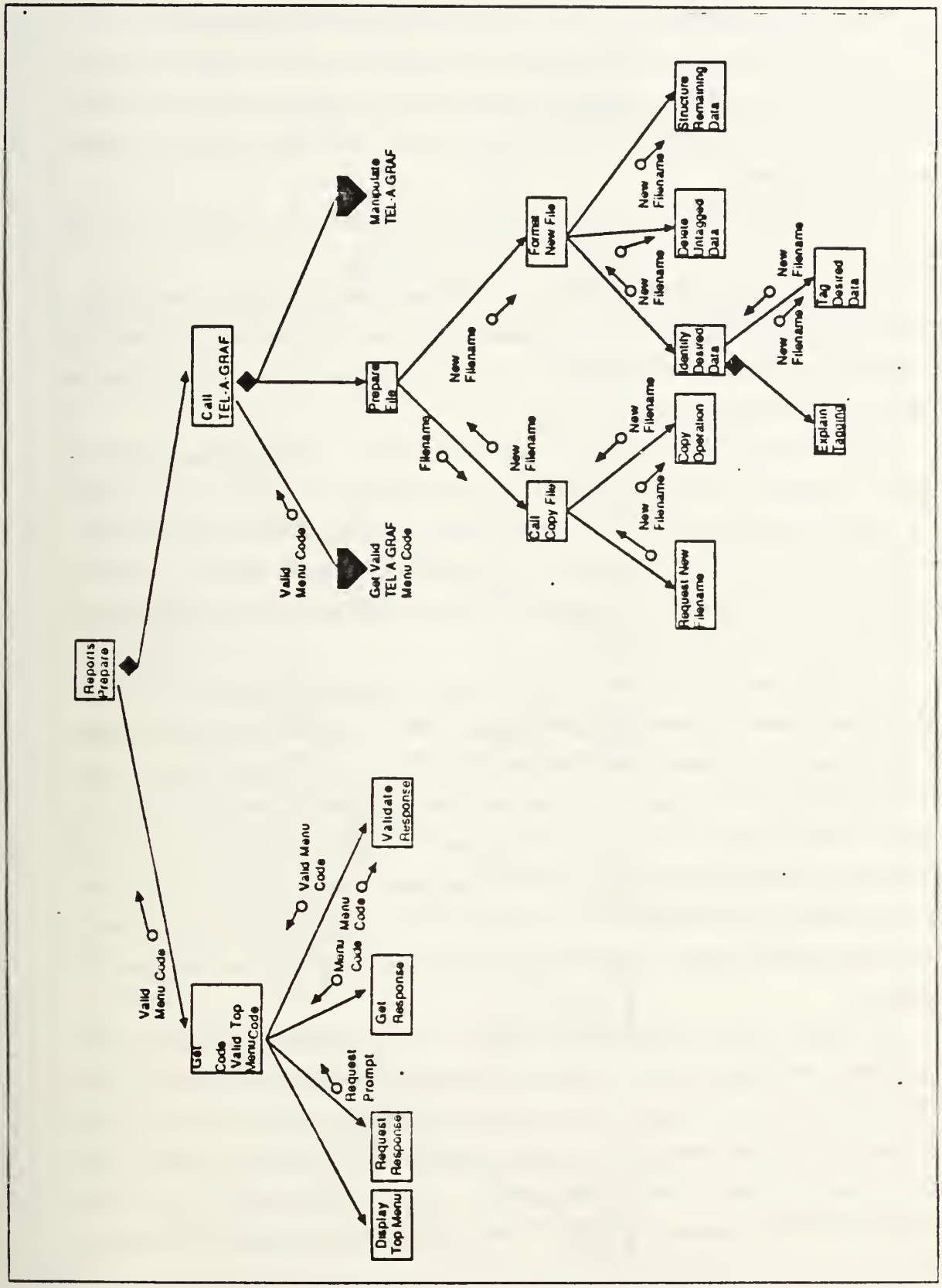


Figure 5.1 Structure Chart of CCA (minicomputer).

(1) *Prepare Reports* (*PR.CPL*). This is the top level program driver of the system. It calls the subprograms to produce the top level menu, accepts and then validates the user's response, and selects the appropriate subsystem or ends the session. Thus, the system is transactional in that the particular subsystem selected is based upon the user's response.

(2) *Display Menu* (*DT5.CPL*). This is the program that displays the top level menu.

(3) *Call Tel-a-graf* (*CTEL.CPL*). The major sub-program of the system, this allows the user the option of selecting standard report formats and nonstandard ones. Depending upon the user's selection, it will call the appropriate sub-programs to implement the desired graphic report.

(4) *Manipulate TEL-A-GRAF* (*MANTEL.CPL*). This program calls the sub-programs that allow the user to select the Cost Center Code, Plot Code, Plot Options Code, and finally open TEL-A-GRAF. The Cost Center Code determines the particular Cost Center to be studied. Plot Code allows the user to select the type of graph he wishes to produce with TEL-A-GRAF. The Plot Options Code controls the level of detail that the user wishes to select.

The modules that select the codes are very similar in structure. We will explain the sub-programs to select the Cost Center Code. The other codes are selected in much the same way. Select Cost Center (SCC.CPL) calls Display Cost Center (DCC.CPL) and assigns the value of the function Validate Cost Center (VCC.CPL) to the variable. That variable is subsequently returned to Manipulate TEL-A-GRAF (MANTEL.CPL). Display Cost Center provides the user a menu of the Cost Centers available for analysis; that number is one for this initial project, but could be easily increased in subsequent versions. Validate Cost Center requests, gets, and validates the user's response.

(5) *Open TEL-A-GRAF* (*OPTEL.CPL*). This determines the data files and include files to be used by TEL-A-GRAF, and calls the routines that actually open TEL-A-GRAF for the user. Data files are those that actually contain the data to be graphed. Include files are essentially programs written in a "structured English" that TEL-AGRAF uses to build the desired graphs. Open TEL-A-GRAF uses the combination of the Cost Center Code, Plot Code and Plot Options Code to make the determinations.

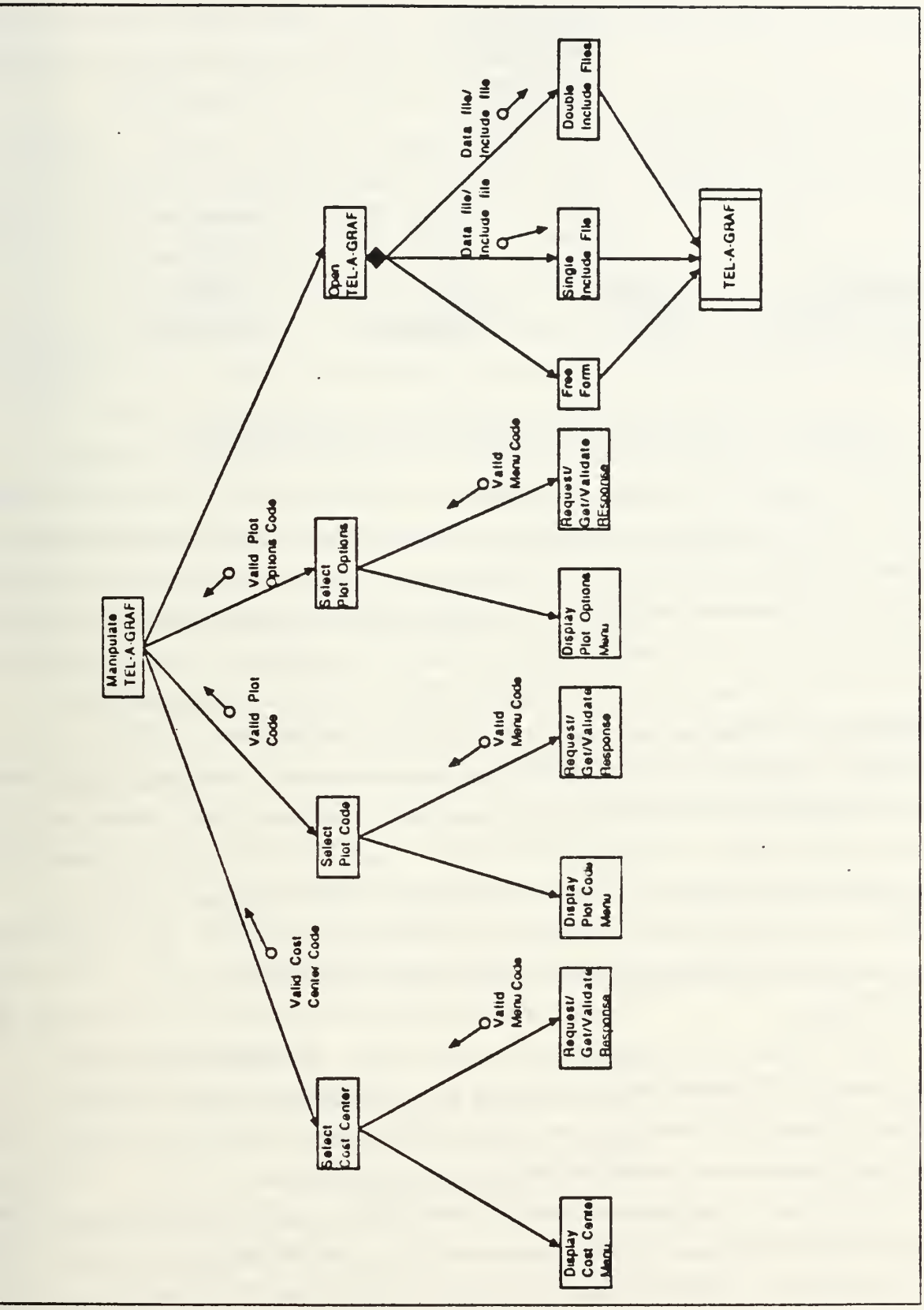


Figure 5.2 Continued Structure Chart of CCA (minicomputer).

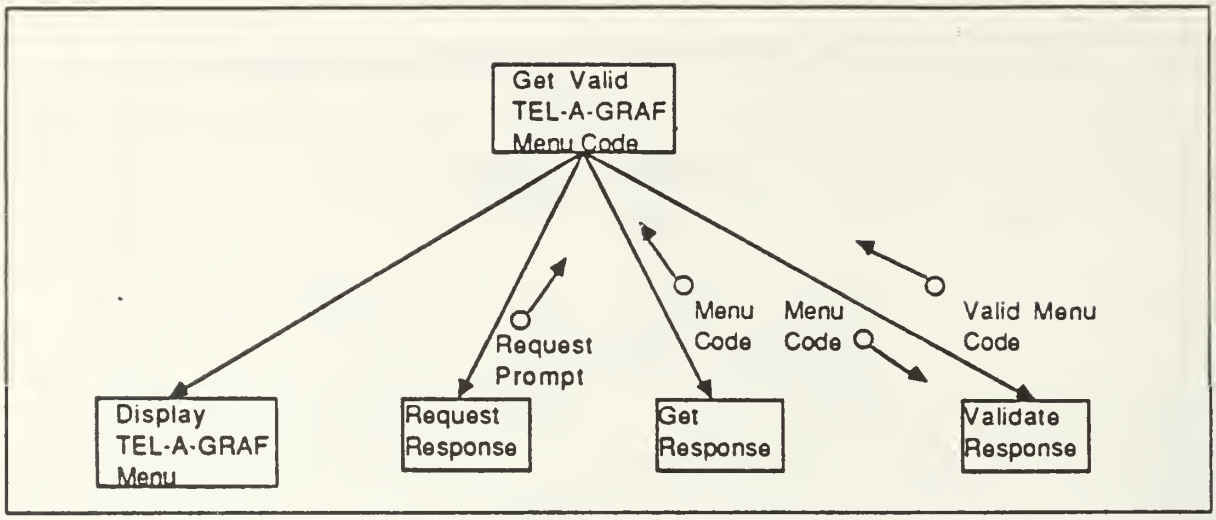


Figure 5.3 Continued Structure Chart of CCA (minicomputer).

(6) *TEL-A-GRAF Interface Programs.*

Six sub-programs actually open TEL-A-GRAF. One allows the user to make free form input to TEL-A-GRAF (FREE.CPL). This would be for those users who had become somewhat familiar with the TELA-GRAF constructs and wish to implement graphs which they design. A second allows the graph to be constructed from one data file and one include file (SINGLE.CPL). The third (DOUBLE.CPL) is used for graphs which have an inset graph, and therefore requires two data files and two include files. The sixth (DOUBAR.CPL) is designed to produce a bar graph which requires one data file but two include files. In order to produce a graph with a bar chart insert and line graph, the fifth (TRIPLE.CPL) is used. (TRIPLE.CPL) requires two data files and three include files, the extra include modifies the plot. The sixth (QUAD.CPL) allows the user to develop a composite graph made up of four subgraphs. It requires four separate data and four separate include files.

b. Usability

This system was designed for users needing an information display capability through high quality graphics. In this case the graphics system's command language was difficult for the average user to learn. The shell provides them with the ability to quickly produce the information displays that they require.

The system is designed in such a way that the users only minimally interact with TEL-A-GRAF. In that way they learn the structure of the command language and gain confidence in their abilities to control TEL-A-GRAF. As they gain

experience, the system allows users to begin to develop their own TEL-A-GRAF command language programs and their own data files.

The advanced user could use the system as a refresher when away from it, or possibly to work out a problem that puzzled him. It is not likely that he would use it to any great degree, however, since it would usually be quicker to go directly into TEL-A-GRAF on his own.

c. Expandability

The structured design of the system allows for easy expandability. Subprograms can be added or deleted, usually with a one line change in the calling program.

The range of changes could include interfacing with additional systems such as a spreadsheet, statistical program, or SABRS. Additional Cost Centers could be added very easily. The only limiting factor would be to provide the data files. This could be circumvented by using a data base management system that would produce the data files, and then an application program which would format them for TEL-A-GRAF. Additional graphic displays could be provided by merely developing new include files and providing the user interfaces to select them.

d. Reliability

CPL is an excellent development environment providing many built-in error trapping routines. It was simple to provide validation of all user inputs requested by the system. The language is also easily applied, thereby further reducing the chances for errors.

The structured approach used to complete the design clearly presents the proposed system. Programming problems were quickly identified and fixed. It allowed us to have a clear idea of what we were doing while coding, much as an outline does for an author. This further protected us from logic errors.

e. Integration of tools

The only tools available to this development effort were TEL-A-GRAF and CPL. TEL-A-GRAF is a very powerful tool, but it is difficult for the average user to manipulate. This situation forced one of us to be dedicated to the development of the required graphs. That effort exceeded our time estimates due to the complexity of the TEL-A-GRAF command language. On the other hand, CPL, a programming language written specifically for Prime minicomputers, exploits the hardware to best advantage.

The integration of the two environments was accomplished by building a shell around TEL-A-GRAF that would allow the average user to make use of menus to produce desired graphs, rather than use the TEL-A-GRAF command language.

f. Problems

Due to our initial lack of understanding of the importance of the data base, this system was built with a file management orientation. It is now obvious to us that this approach was in error. The system should have been built around a data base management system. This would have produced a more flexible system.

Geographic separation from the users during the development increased the difficulties. Close coordination with the users was then not possible, and resulted in misunderstood requirements and goals. We only had access to the development language via networking over FTS lines, which at times became unusable due to noise. In addition, the version of TEL-A-GRAF locally available to us was not directly compatible with that available at the user site. Consequently, last minute changes were required before demonstrating to the users.

A major limitation of CPL is that sub-routines can not pass parameters up to a calling procedure. Thus most "atomic" routines had to be programs. This resulted in many small programs making up the system. Although this was not a problem for us, it could be a problem for the users and those who will maintain the system.

2. Cost Center Analysis (microcomputer)

a. Introduction

This system was developed to address the most critical problem of the minicomputer implementation, the lack of a data base management orientation. In order to develop a system that allowed easy data maintenance, it became apparent that a file management application would not be effective for an historical data base. This version is integrated with a DBMS in order to provide the easy expandability which would be beneficial to the users.

Originally, this was to be a concurrent development of the Cost Center Analysis system. Therefore, there was no consideration given for compatibility of the "micro" development language with the "mini" development language, C and CPL respectively. They were to address two separate development issues, and the language used for each was irrelevant to the efforts.

The design of the data base is relevant to each since it could be applied to either system. Oracle was therefore selected as the DBMS because it is available for

microcomputers, minicomputers and mainframes in general, and in particular is scheduled for implementation at the shipyard on the Prime network. This will provide the users greater flexibility.

b. Requirements

Cost Center Analysis (microcomputer) hardware requirements are an IBM PC/XT/AT with at least 640KB and a hard disk. A printer is optional for the output print options.

The software requirements are the Oracle Data Base Management System (DBMS), PC/MS-DOS, and the Cost Center Analysis and Graphic Utilities programs, all installed on a hard disk.

c. Module description and functions

The hierarchy chart depicted in Figures 5.4 and 5.5 graphically describes the interrelationships between the following program modules.

(1) *Cost Center Analysis.* This main module contains the three major sub-modules of the system:

1. Cost Center Information
2. Command Level Entry
3. Graphics Displays

It provides the interface for Oracle and graphics. It also allows easy access and display of Oracle.

(2) *Graphics.* This module allows the user to use already developed graphs with data not obtained from Oracle directly, but input to a file by the user. Oracle can also read specified data into a file form within the Cost Center Information module. This data can then be plotted from within the graphics module.

(3) *Command Level.* This allows the user the opportunity to use Oracle at the command level, through the User Friendly Interface (UFI). Ad hoc queries, updates of the data base, deletions, insertions and other procedures can be accomplished in this mode.

(4) *Cost Center Information.* This is the main menu driven shell for CCA. It allows easy access to specified information and display of that information. CCI also sends specified data to a file for the graphics routines to use.

(5) *Budget VS Expenses.* This allows display and comparison of budget and actual expense information by various categories, and interfaces data with graphics for further displays.

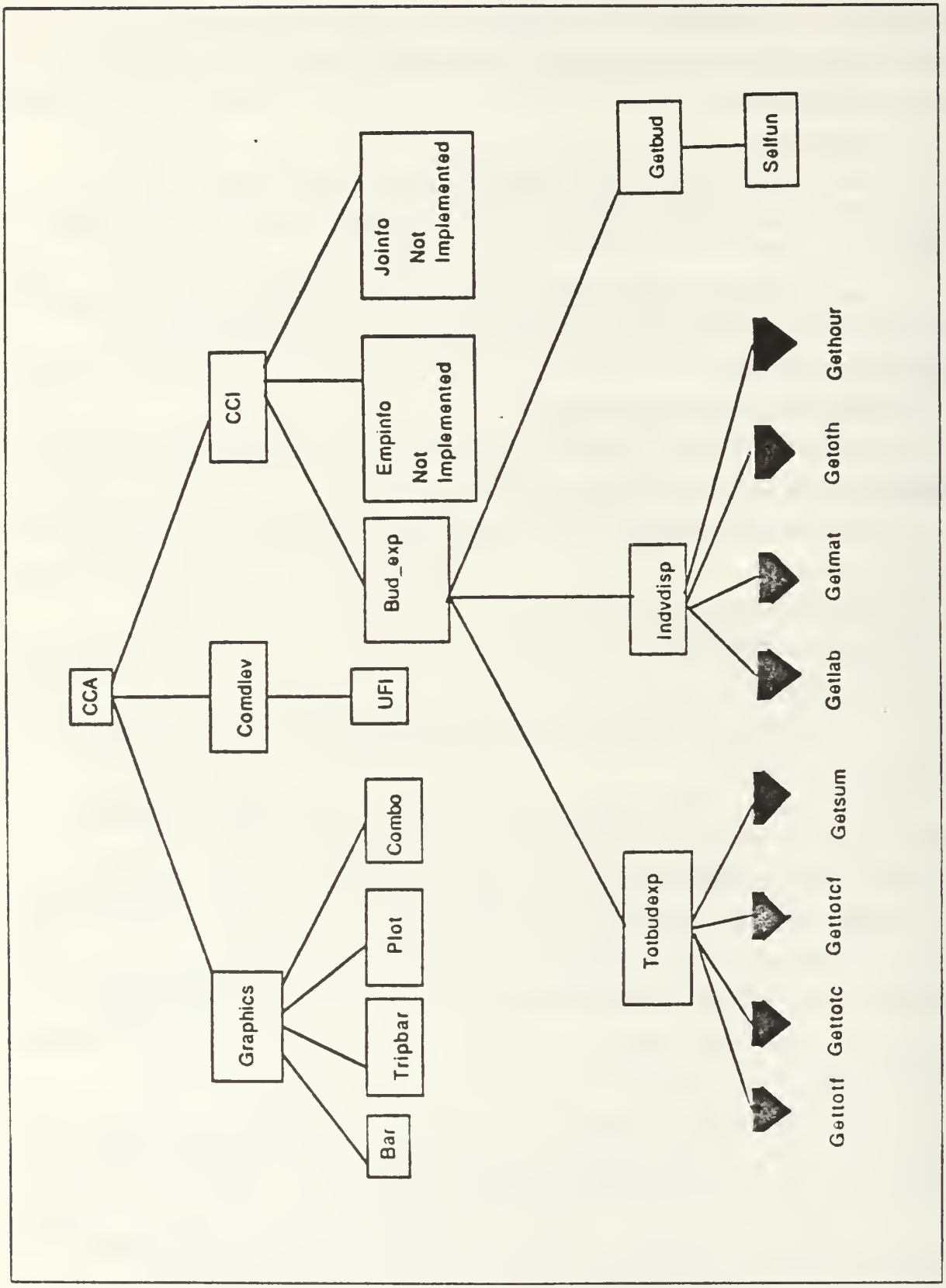


Figure 5.4 Hierarchy Chart of CCA (microcomputer).

(6) *Individual Display.* This module displays budget vs expenses to date in thousands of dollars for either Labor, Material, or Other, sorted by Cost Function/Cost Class.

(7) *Budget Summary.* This displays budget by Cost Function/Cost Class for the current fiscal year.

(8) *Display Labor.* Budget vs expense by Cost Function/Cost Class for Labor are displayed.

(9) *Display Material.* Budget vs expense by Cost Function/Cost Class for Material are displayed.

(10) *Display Other.* Budget vs expense for Other costs by Cost Function/Cost Class are displayed

(11) *Total Budget VS Expense.* This module sums Labor, Material and Other for budget and expenses to date by Cost Function, Cost Class, Cost Function/Cost Class and Cost Center as requested, and sends the information to the graphics utilities when directed.

(12) *Total by Cost Function.* This sums budget and expenses to date by Cost Function and displays. It sends the information to a file for graphing, upon request.

(13) *Total by Cost Class.* This sums budget and expenses to date by Cost Class and displays. It sends the information to a file upon request for graphing.

(14) *Total by Cost Function/Cost Class.* It sums budget and expenses to date by Cost Function/Cost Class and displays.

(15) *Total by Cost Center.* This module sums budget and expenses to date for the entire Cost Center and displays. It sends the information to a file for graphing upon request.

d. Implementation

The CCA system (microcomputer) is operational but not to the extent that we originally hoped. The memory limitations of the PC prohibited us from linking graphics with Oracle directly. To perform the graphics, extra commands must be initiated at the DOS level. This was not the original intent. The system indirectly links Oracle with the graphics, which still accomplishes our original goal.

The degree of difficulty in using C to drive both the graphics and Oracle was underestimated at the outset of this project. Another limitation had to do with the disk storage space needed to support all the products that were tied together. The

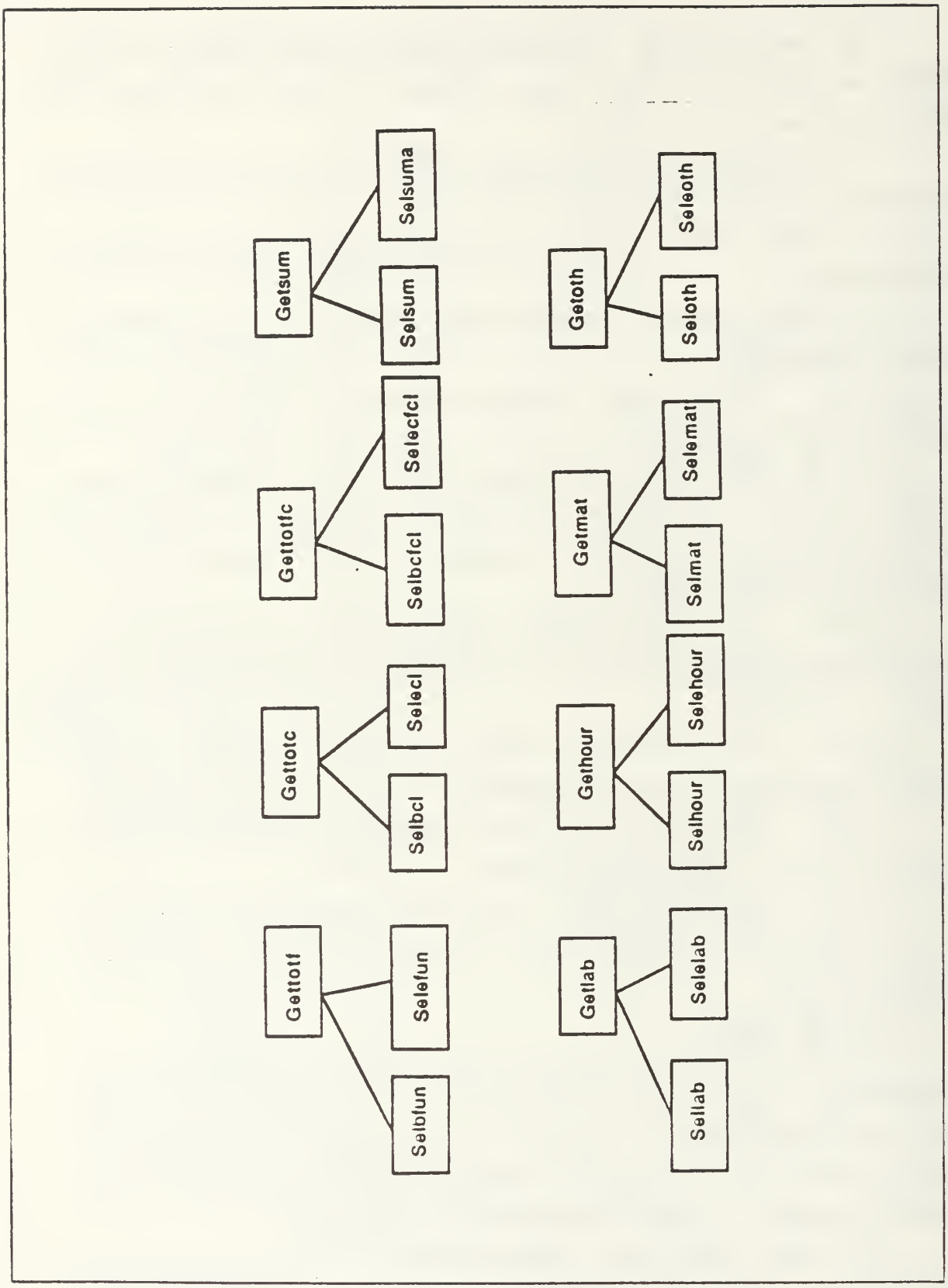


Figure 5.5 Continued Hierarchy Chart of CCA (microcomputer).

system we developed this on was already heavily loaded with other systems and thus hampered our development. This initial prototype system will allow the user a chance to see what a final system could do, and to further direct efforts in the development of future prototypes.

e. Usability

This analysis system was primarily designed for the inexperienced or casual user. They would be individuals, such as managers, who have not yet learned the Oracle command language, SQL, or never intend to learn it.

During the analysis portion of this project we identified the information needs of the users. Queries were written within the "shell" to specifically address these needs. So, the average user is not likely to require ad hoc queries, at least in the short term.

However, we have included the ability for them to handle unexpected or unanticipated questions, or to make necessary data modifications. This ability is provided by our system through the Oracle UFI. This interface does require using the command language, since the user will no longer have the menu support of the shell.

Advanced users who would tire of the system menus could enter the UFI through the top-level menu. Occasionally, they might make use of the system, but if they know the command language they will probably enter the UFI directly.

f. Expandability

As mentioned, ad hoc queries are supported through the use of the command language option. The Command language allows the user to interface with Oracle via the UFI as described above. The user's manual (Appendix C) shows many examples and possible approaches to retrieving information using UFI.

This system was designed to be used by only one Cost Center. However, this is a prototype system that could be expanded to all Cost Centers within the shipyard. To accomplish this, data base views of each Cost Center could be developed, giving the manager of a Cost Center access only to his own information. Higher management could have access to all data as necessary. Only minor adjustments would be necessary to implement this, namely inserting another module in the hierarchy above the CCI module allowing the specification of a Cost Center, combination of Cost Centers, or all Cost Centers depending on the access level of the requestor and his interest. The specific views can be provided through the data base management system. The amount of data necessary to support just one more Cost

Center would, however, double the amount of data. This would not only increase the storage requirements, but would also increase the information retrieval time.

g. Reliability

Any large program written in C is suspect when the question of reliability comes up. The strange and wonderful things that C programs can do when errors occur can be truely awe inspiring. However, painstaking error traps have been built in to counter all known problems. Each input from the user is checked for validity. Numbers are checked to see if they are in range and if not, the user is returned to the same menu. The user also has the opportunity to review his inputs to insure that the value inputed, was the value he really wanted.

B. GRAPHICS MODULE

After we constructed the initial structure chart we exploded both main modules: the control module and the graphics module. Then we began the detailed design of the graphics module by following the ROMC approach [Ref. 2]. The initial representations were given to us by the user, who had been creating graphs for his own use on a microcomputer using LOTUS 123. These graphs were time consuming to produce because the data had to be extracted and inputed by hand.

The user wanted multiple graphs on one page which would summarize the data more effectively. The first graph that he wanted was a bar graph of the budgeted amounts for the Cost Center, broken down by Cost Function. On the same page with that graph he wanted a plot of the budget versus the actual expenses as a function of time (see Figure 5.6). This plot was applicable to either the total Cost Center or for one of its Cost Functions.

Our methodology for designing the graphs paralleled the methodology we incorporated in our DSS development: an iterative approach. These graphs were then scrutinized and revised as necessary.

Most of the actual keying of the code was done 180 miles away from the shipyard via telephone lines. The original coding iterations were done on the Naval Postgraduate School's IBM 3033, before entering on the Prime. The School, like the shipyard, has TEL-A-GRAF and the code is transferable between systems, with one exception. The window sizes created for the graphs on the IBM were slightly different for the Prime. Therefore, complete testing could not be performed until we actually went to the shipyard. We could test for coding errors, by observing a run, but the terminal we used to input the code did not support the graphics.

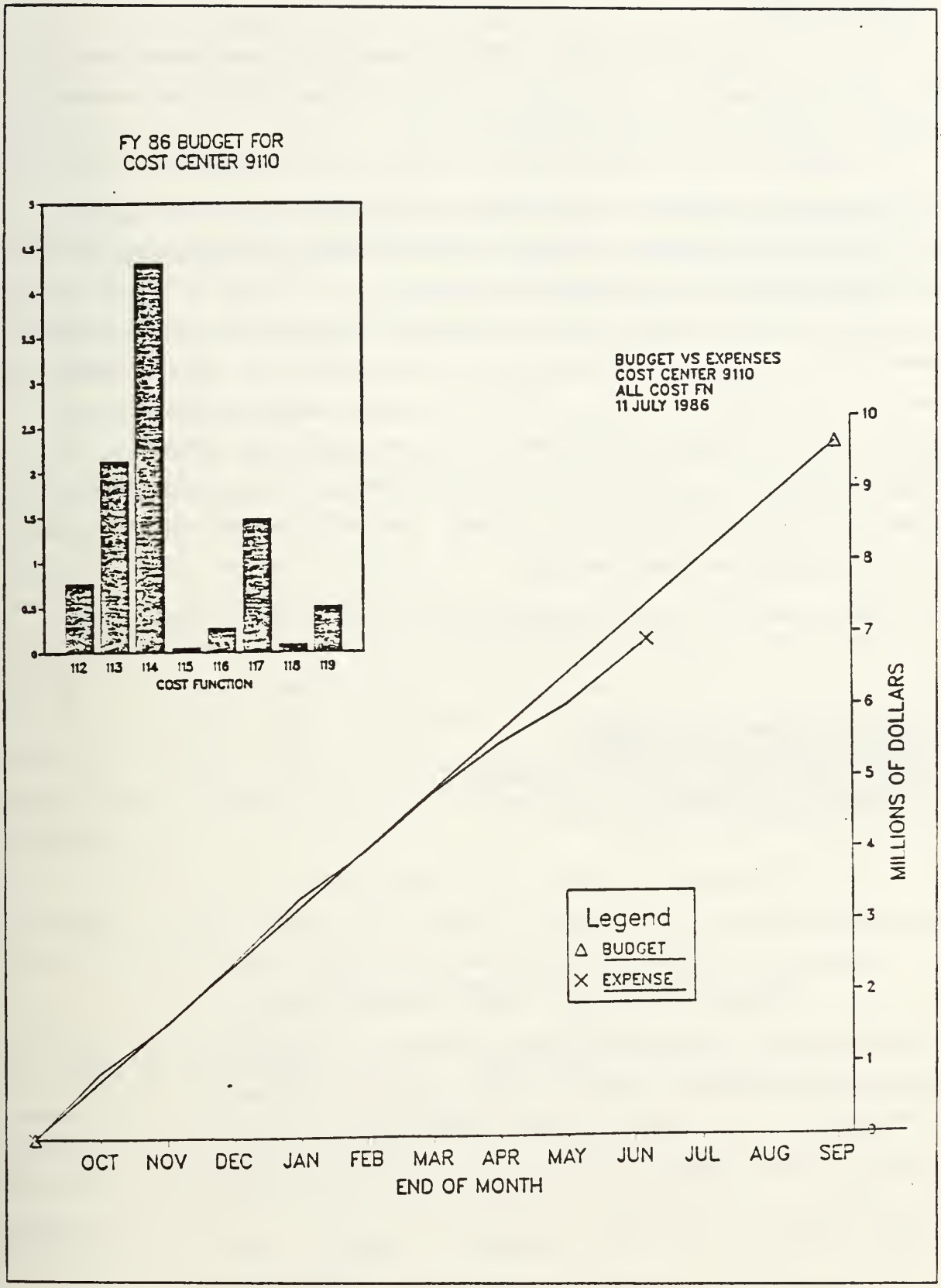


Figure 5.6 Composite Graph, Bar Chart and Plot of Budget VS Expense.

1. TEL-A-GRAF

As our first step we learned TEL-A-GRAF, the graphics system that we would be using. "TEL-A-GRAF is a conversational computer graphics system that produces publication quality charts and graphs [Ref. 12: p. A-3]."

The tutorial for TEL-A-GRAF [Ref. 12], walks the user through some simple graphs. The command language is English-like and was easy to follow for the simpler graphs. TEL-A-GRAF encourages the user to experiment when building graphs until he gets the desired format; again an iterative approach.

A profile file tells TEL-A-GRAF which device the user is on, what his secondary device is, and other facts necessary for TEL-A-GRAF to operate. If the user does not have a profile file, TEL-A-GRAF will prompt the user for the necessary information. For our system we wanted the control of TEL-A-GRAF to be maintained by the control module. Therefore, a profile file must be established prior to the use of this system for each user. A Sample profile file that was used when developing the system is shown in figure 5.7 .

```
PRIMARY DEVICE IS TEX618.
SECONDARY DEVICE IS POP.
SECONDARY DEVICE UNIT NUMBER IS 1.
PAGE LAYOUT IS HRV.
ERROR REPORTING LEVEL IS 0.
```

Figure 5.7 Sample TEL-A-GRAF Profile File.

TEL-A-GRAF can be used on a variety of devices. The primary device refers to the main device that the user wants to create his graphics on. When the commands "GO." or "DRAW 1 2." are issued the graphics file is created for the device named as primary device. If the command "SEND" is issued the graphics file created is for the secondary device.

TEL-A-GRAF also has the capability of creating a device independent graphics file that can be used later on any graphics device, such as printers, plotters, or

other graphics terminals. This is accomplished through the use of the Post Processor, known as "POP". By naming either the primary or secondary device as "POP", a device independent file will be created. This file can then be executed later using the DISSPOP command. Appendix B provides further information on the use of TEL-A-GRAF.

The profile file can be changed while in TEL-A-GRAF at the command level. When the generate prompt of TEL-A-GRAF appears, a device or value of a profile file can be changed by issuing a command such as "PRIMARY DEVICE IS POP." This command would change the primary device to the post processor for the remainder of that session.

a. Composite Graph

Once we were familiar with the TEL-A-GRAF language, the actual coding of the first graphs began. This was in itself an iterative process. The basics learned from the tutorial [Ref. 12], and the specific requirements for these graphs were aggregated to produce the finished product. Figure 5.6 shows the final graph for the two initial graphs suggested by the decision maker. One of our main goals while developing this code was to keep the graphics module independent. We designed the graphics to run from a batch type mode. We assumed the data was in independent files in the necessary format. This allowed us to concentrate on the display and not worry at this point on how the data got to be in those files. This is a flexible approach that will allow the shipyard to choose any data base system to complete the implementation.[3]

Each graph required a data file and at least one include file. The data file contained the formated data as TEL-A-GRAF would accept it. Figure 5.8 shows one format that TEL-A-GRAF will accept. Other formats are discussed in Appendix B.

The include files contained the TEL-A-GRAF commands which set up the graphs. The original include files contained the titles and labels for cost center 9110. To change the title or labels to create a similar graph, a secondary include file was created. This secondary include file changed title and label line commands in the major include file. This allowed the flexibility to create the same graph from different data. These include files could also be called up by hand and changed line by line to fit a different need of the user. This made the design of a modular system simpler. The first

[3]We recommend a relational data base for flexibility. The data will be discussed in detail later.

```
INPUT DATA.
"BUDGET"
 0 0 1 0.8035 12 9.64198
"EXPENSE"
 0 0 1 0.92 2 1.59901 3 2.4567 4 3.34567 5 4.0002 6 4.78999 7 --
5.477 8 6.008 9.2 6.91127
END OF DATA.
**FILE**
```

Figure 5.8 TEL-A-GRAF Data File.

include file contained the commands for a graph labelled for Cost Center 9110. If the graph was to display a Cost Function, instead of the Cost Center, another include file was needed to change the labels and title. When other Cost Centers are added to the system, the same process will apply.

b. Triple Bar Graph

Once the initial graph was complete, we began the design of a new representation. Based on our analysis, we determined that a bar graph which displayed the total budgeted data, the budget as a straight line percentage of the elapsed fiscal year, and the actual expense to date would be useful. This is basically the same information displayed in the original composite graph. However, we felt that this prototype should offer a choice of representations to the manager. One purpose of a DSS is to provide the decision maker with the appropriate information in a format with which he is comfortable. This allows the decision maker to choose one of the formats.

The same iterative process was followed when creating the second graph. After repeated testing and manipulations, the graph was ready. This graph was also created using the TEL-A-GRAF include files. This allowed for one basic command file that the other includes build from. This strategy made the manipulations of the labels only a matter of changing one line, rather than rewriting the entire program. The data was again assumed to be already in place, properly formatted.

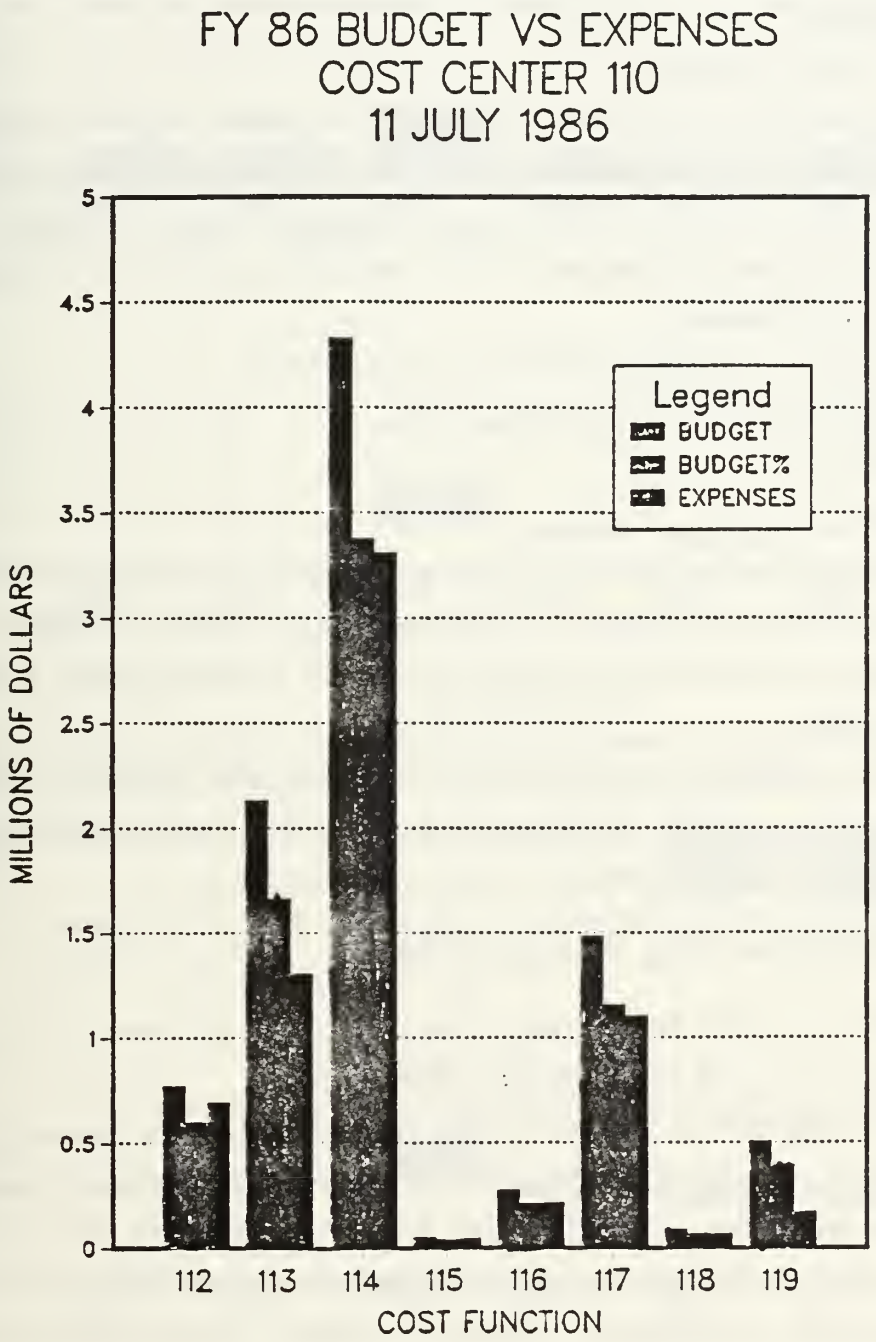


Figure 5.9 Triple Bar Graph for Cost Center 9110.

Figure 5.9 shows the finished graph. This graph summarizes and compares the budget to expenses, both to the total budget and to the percent of budget as a function of time. Although not specifically addressed, variances can be identified and estimated using this graph.

c. Variance Analysis Graphs

The last graphs developed represented four bar graphs displayed on one page. These graphs display variance of the Cost Center for overtime, straight time Hours and Labor, Material, Other and Total. Again, the same iterative approach was taken in coding. The data was also assumed to be present in the data file. The graphs contained the following information:

1. Percent Expended
2. Data Normalized on the Percent of Elapsed Time
3. Variance in Dollars
4. Percent variance

These graphs were suggested by the user. They are designed to fill a gap in the analysis of variance. The exact manipulation of the data to attain these figures is explained below. Figure 5.10 shows the structure and format of these graphs. A brief description of each follows.

The percent expended shows the percent of elapsed time based on the date of the data and the amount of the fiscal year elapsed. The remaining percentages represent the percent of the budget expended. The formula for this is:

Percent Expended = Expense / Budget

Labor is broken down into overtime and straight time. These are key areas of interest to the manager because he usually has direct control over Labor.

The data normalized on percent elapsed time, normalizes the elapsed time to one. This normalization changes the percent expended from the first graph into a percentage of elapsed time. That is, if 20% of the budget was expended and 20% of the year had elapsed, the normalized value would be one, the same as elapsed time. If the normalized percentage is less than one, less has been spent in that category. If the percentage is greater than one, more than the percentage was spent.

Normalized = Percent Expended / Percent of Year/ 100

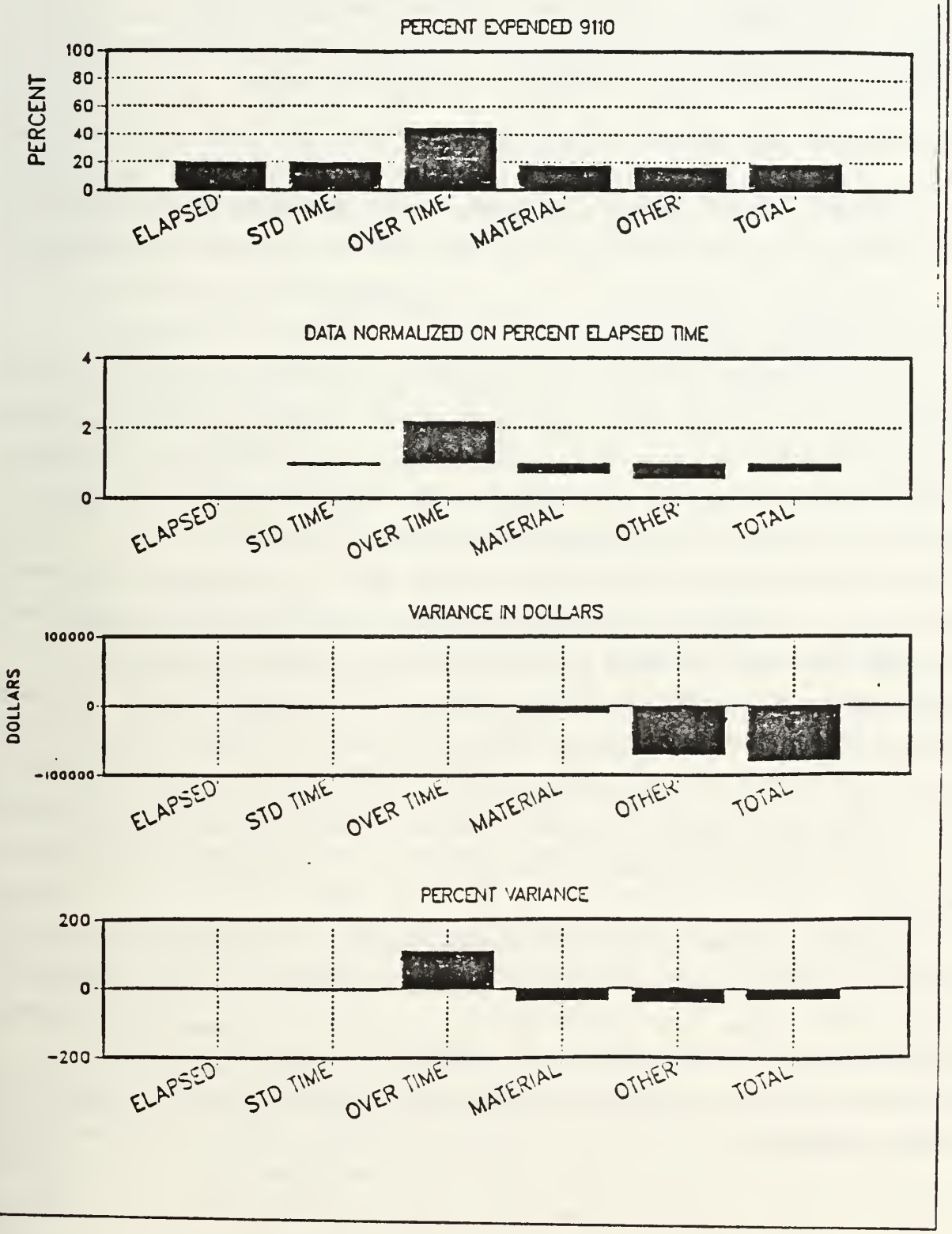


Figure 5.10 Four Graphs for Variance Analysis.

Variance in dollars gives the dollar amounts of the variance. This is important because the percentages can be deceiving. If a category has a large variance, but only a small amount of money was budgeted, the dollar amount may be insignificant. For items with large dollar amounts, small variances could involve large sums of money and be much more significant to the financial situation.

Variance= Expense - (% of Year * Budget)

Percent variance displays the values as the percentage based on the budgeted amount. This is the percent of the budget divided into the amount expended.

% Variance = Variance / (% of Year * Budget / 100)

2. **Graphics on the Microcomputer**

The implementation on the microcomputer followed the same design as the implementation on the minicomputer for the graphics. The variance graphs, however, were not implemented on the "micro." The graphs produced and the completion of the data base on the microcomputer, demonstrate the technical feasibility of this implementation. However, there are limitations that must be considered.

The memory limitations on the micro did not allow us to implement a fully interactive system. The graphics package used, GraphiC, combined with the data base programs and the control programs would not run as an integral system due to insufficient memory. This was resolved by having the control program interact only with the data base. The control program would ask the user if he wanted a graph of appropriate data. If the user responded positively, a data file would be created that would be accessed by the graphics program. The user would then leave the control program, choose the correct program to run, and run it. The data file is accessed automatically.

The graphic programs on the microcomputer were more complicated to code than the TEL-A-GRAF graphics on the minicomputer. Compiling and linking slowed down the iterative design process, but the same procedures for developing the microcomputer graphics were followed. But formatting the output data was not a problem since the data base management system and the GraphiC utilities were written in a common language, C.

C. DATA

The data was not a major concern for us during the first iteration with the minicomputer, except for identifying a particular data element needed for each graph. Since the shipyard was responsible for extracting the data, we concentrated on the control program and the graphics. When we began the microcomputer implementation, the overall design was completed. However, this implementation actually dealt with the data, so it became the central focus. Although we side stepped the issue in the initial effort, we found that the data structure played a key role in the design.

1. Data Base on the Microcomputer

Originally, we identified the origin of the data elements. The origin of the data was the reports that were generated on the shipyard's mainframe. However, halfway through the analysis, the reports changed. The new reports from SABRS contained the information in a variety of formats and was produced on the minicomputer. This report (SBR-22A) summarized the data by Cost Function/Cost Class for each Cost Center. This was the best way to store the data for our data base because all combinations of Cost Centers, Cost Function, and Cost Class can be derived from this information, thus minimizing the storage of the data.

One problem occurred that caused our data base to be larger than originally planned. Entries that do not have values must be entered as zero. This greatly expanded the data base because many authorized Cost Functions rarely use some authorized Cost Classes. However if they are omitted, inconsistencies occur when the budget and expense tables are joined in an operation. For example, if a particular Cost Function/Cost Class did not have a budgeted amount, but did have a later expense, the expense would be lost in a comparison. The tables of the data base are joined on the Cost Function/Cost Class combination and if there is no value for a particular Cost Function/Cost Class, the value from the expense table is not included in the resulting joined table.

The result is that an entry must be present for each authorized Cost Function/Cost Class. The budget table contains only one value for each authorized Cost Function/Cost Class. The expense table on the other hand, must contain a value for each update. This means the table increases in size on each update by a constant. If the update is made every two weeks, approximately 26 updates will have occured. That means the expense table will be 26 times the size of the budget table.

2. Historical Data Base

If an historical data base is desired for the DSS, an additional field will need to be added to the budget table: the fiscal year. All selects based on the current fiscal year will have to be identified by year, or by maximum fiscal year, if the next year's budget is not yet installed.

3. Data Base Design

We anticipate that the final system's data base will be electronically updated. The volume of the data is very predictable. The expense data will be appended at mid-month and at the end of each month.

The data retrieval rate is estimated to be fairly low 10 to 20 times daily. At critical periods in the fiscal year, the utilization will be much higher, such as the end of quarters, prior to mid year review, and at the end of the fiscal year. The retrieval rates at those times is estimated to double or triple.

Managers can allow others access to their data base with read only privileges. Some information and views can also be restricted using the data base's userids and passwords. This can increase the utilization of the data base while still maintaining control over the dissemination.

a. Bachman Diagram

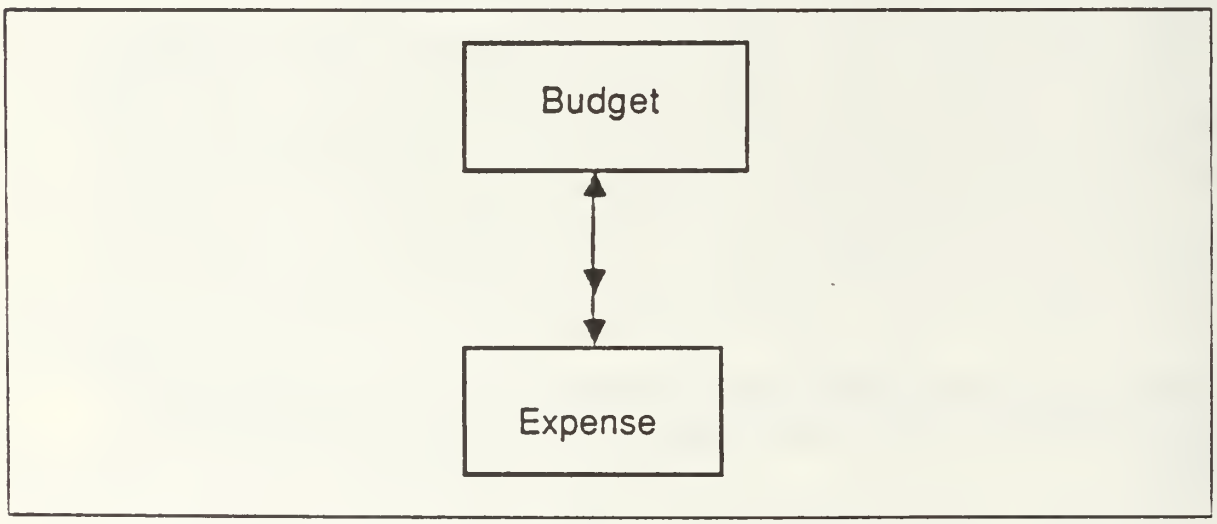


Figure 5.11 Data Base Design Bachman Diagram.

The main relationship for this implementation is the one between budget and expenses. As depicted in Figure 5.11 this is a one to many relationship. Even if

budgets are developed monthly or quarterly the relationship holds. The newer budgets simply supersede the older ones.

b. Relational Model

The record structures are depicted for the data base design (keys are italicized):

1. BUDGET (*COST FUN NO*, *COST CLASS NO*, ST HOURS, OT HOURS, ST LABOR, OT LABOR, MATERIAL, OTHER)
2. EXPENSE (*COST FUN NO*, *COST CLASS NO*, DATE, ST HOURS, OT HOURS, ST LABOR, OT LABOR, MATERIAL, OTHER)

c. Normal Forms

Normalization is a process by which we attempt to minimize "anomalies" in the data base design. These anomalies generally can cause data inconsistencies, loss of entire records during updating, and inappropriate relationships between different record types when joining tables. The effort is to normalize to as high a degree as possible, trading off retrieval performance and increasing interrelational constraints [Ref. 13].

In the discussion of normal forms, we do not attempt to justify our normalization beyond the third normal form. We felt that the excessive interrelational constraints would tend to make the data base less workable. In addition, all tables are in first normal form, because of the lack of repeating groups in our data structures, so discussions that further illuminate that point are not needed.

(1) *BUDGET Table.*

This table is in second normal form since all the non-key attributes rely on all of the key (Cost Function number and Cost Class number). Additionally, all the non-key attributes are independent of each other. For example, Hours (straight or overtime) does not directly relate to Labor cost, since it requires computations and reference to other schedules in order to be produced. That is the reason why both Hours and Labor are listed. With the independent non-key attributes, there are no transitive dependencies, placing Budget in third normal form.

(2) *EXPENSE Table.* The Expense table is similar in structure to the Budget table. However, it has an additional attribute in the key, Date. All the non-key attributes require reference to all attributes of the key, so this table is also in second normal form. Again the non-key attributes do not directly relate to each other, so it is also in third normal form.

d. Interrelational Constraints

There are going to be a certain amount of interrelational constraints in any data base. The important questions to consider are do the designers realize its existence, do they understand why it is there, and are they in control of it. In this data base the existence of the interrelations is largely a function of the nature of the problem that the data base attempts to address. The greatest interrelational dependency is caused by the Cost Function and Cost Class numbers. They are the most integral pieces of data within the data base. Without them the tables that contain them would be meaningless. They are the language for tracking costs of operations within the Cost Centers of the Shipyard. They categorize the data.

Combining all the tables with these common attributes is not a sufficient answer to reduce interrelational constraints, since that would reduce the normalization of the tables, and increase redundancy. The trade-off is to keep this interrelational constraint, since it requires the least overhead and maintenance.

VI. INTERPRETATION OF THE DEVELOPMENT EFFORT

A. ANALYSIS OF THE METHODOLOGY

The approach we applied to this development effort appears to have worked well for this project. The requirements presented to us were not clear, since the users did not have a concrete idea of what they specifically needed or wanted for this information retrieval and display system. In addition, we did not have a clear understanding of what was being asked of us, and what the effort would entail.

The methodology we selected forced us to be thorough in our analysis and design, and gave us flexibility to alter development directions. We elected to follow a combination approach, which we felt would best answer the needs and requirements of this particular project. This pilot project was carried out within the framework of a prototyping approach. The development environment follow the dictates of Yourdon for a prototyping environment [Ref. 3: pp. 225-226]:

1. There is only a single user or at most a small group of users who are 'localized' in the sense that they work in the same organizational group and within the same physical location.
2. The data model exists or can be easily created.
3. The application is small to medium.
4. Everyone agrees that the prototype is only a 'toy' system and that it is intended as nothing more than a model of the production system

The prototype approach was essential. We were attempting to develop a system based on an unclear problem statement and no prior experience; we required the slack that this approach could provide. Meeting Yourdon's premises further supported the prototyping decision.

Our approach to prototyping was not simply to go forward and start writing lines of code. Within this framework we applied structured analysis to conduct the analysis of the present situation. We selected the structured tools in order to provide ourselves with the clearest appraisal of the users' present system. The structured design following the analysis easily flows from the same tools and provides the users with a clear documentation of how the system was designed. The resulting structured specification of data flow diagrams, data dictionary and structure chart clarified our understanding and assisted the coding efforts of the two phases.

1. **Documentation**

The program design of the first project more closely followed the structured design than the second did. The major reason is that during the second, the orientation of the project shifted. The importance of the data base design became apparent during the microcomputer implementation. The design and structure documents are purposely made broad to keep from locking in on one system. Although the constraints of hardware and software greatly limit the alternatives, by keeping the design flexible and general, several alternatives became apparent.

In actuality, we built two prototypes in this project. Although neither are complete in the sense of a production model, both contributed to the overall project. Neither system as yet has the capability to be updated electronically. Yet both rely on the concept of the electronic update based on the SABRS reports for their feasibility. The minicomputer version has a data base in design concept only. The microcomputer version does not have all the graphs implemented, and none of the variance analysis graphics. However, although they are different systems, together they complement each other by providing us with differing views of the same project.

The quick implementation of the microcomputer version proves that the design was generic enough to be implemented on two completely different systems. The design for the major modules was already completed. We quickly found that a more detailed design was necessary to actually develop the data base. This is part of the iterative process. Now the data base design is completed and could be implemented on the minicomputer relatively quickly.

All the modules were designed to be independent, with low coupling. This helped us create a system that could be implemented on two different systems. As the final alternative is selected, only the automation boundaries and the timings of the system will probably need to be changed in the design documentation.

2. **Iterative Approach**

The iterative approach also was very successful for us. Allowing the user to see the progress and make changes throughout the project enhanced the communications between the user and the developers. This communication is very important to any methodology and no less with this one.

The iterative, prototyping approach proved its flexibility when the users changed to a new accounting system, known as SABRS. SABRS changed the format of the input data for CCA. Although not major, the changes demonstrated the significance of built-in flexibility in a development effort.

The point of this discussion brought out by the structured techniques is that the final system, whether prototype or production, must be maintainable. By this we mean that it must be easily adapted and modified to meet the changes during the iterative approach and changes in user requirements. Therefore, we strove to ensure that the documentation of our prototype was as clear and as understandable as possible.

3. **Communication**

Communication is not directly related to any methodology, but is a key factor in any analysis. An analyst must be able to communicate with the user. During our analysis phase, we conducted several interviews. After focusing on a decision, budget control, we set up appointments to meet with various the budget experts within the shipyard. At this point the project scope had not been fully defined and we were looking for general information and procedures on the budgeting process. As system analysts in an interview situation, we quickly found ourselves on the receiving end of several questions. Instead of doing the interviewing we were being interviewed. We lost control of the initial interview and, even though they were still cooperative, we initially lost the interest of a potential user of a DSS.

The lessons learned from this interview were many. Interviews, especially initial meetings, must be carefully planned. A brief summary of the questions, or the type of questions that we were going to ask presented to the user prior to the interview would have been a much better way begin. The problem definition should have been defined better prior to the interview. The Term DSS was also used freely, which brought connotations of wonderful systems that have all the answers. Our system was a pilot project, not a production system, and this fact should have been introduced up front.

We found that our interviews with the users were tainted by our unfamiliarity and our own preconceptions of what they wanted. These problems were largely an outgrowth of our inability to ask the right questions, our lack of experience with the interview process, and in some cases our incomplete technical knowledge. Generally, our abilities and interviews grew as our understanding of the system and our familiarity with the users grew. The best interviewing technique that we found was knowing ahead of time what to ask, and asking it in a manner that does not cause a defensive response.

This is an area that is usually glossed over by most authors in the DSS arena. Possibly most authors do not perceive a need. One answer might be that most developers do not think they will make those "classical types" of human interaction errors. To many, these issues may seem of little consequence. Finally, these development efforts are completed by information specialists and not clinical psychologists. Therefore, the tendency is to deal only on a cursory level with what is not very tractable.

4. **Remote Site Development**

Not being able to be on-site during the development process was definitely a liability. This fact more than any other slowed our progress. It was difficult to find time to consult with the users, and the distance of the commute precluded conferring with the users over what we perceived as smaller matters.

On a couple of occasions those smaller matters were a lot more important than we had thought. Additionally, a better job of analysis could have been done if we had more opportunities to consult with the users. For example, the perception of being considered outsiders to the organization might have been reduced. We would urge any would be developers to spend a significant and consecutive block of time with the users if at all possible.

B. RESULTS FROM CCA PILOT PROJECT

Based on our methodology and the resulting prototypes, several conclusions and alternatives can be drawn. The first area of concern is the data extraction. The data for the system must be extracted from SABRS. This extraction system is the key for continuance of alternatives based on this project, other than to stop any further development at this time. The worth of the system has been demonstrated, making this option unlikely. Once the data base is extracted a repository must be set up.

The data base approach is essential to this type of development. A file management approach is too limiting and unwieldy for systems that frequently update information. In a file system, either data files must be overwritten with new data, or if new files are added with each update, the application program must be altered for each new file. A pure file system of data management is useful only for a test system, not for a "real world" system.

Therefore, the kernel of an information display system should be a data base management system (DBMS). This requirement also applies to DSS's. The DBMS

reduces the amount of coding and design that the developers must use to build a DSS. The capabilities of the overall system can be quickly enhanced without the design and programming complexities. Some of the desired capabilities would be a sophisticated command language, documentation utilities, easy data loading and deletion utilities, user control, and security utilities. The flexibility of the DBMS is an outgrowth of the degree of capabilities it possesses. Developers only need to design interfaces to the DBMS.

1. Data Base Design

When designing an information system, a data reference is critical. The most flexible data base system that we recommend is a relational data base. A relational data base can offer several advantages as demonstrated in the microcomputer implementation. A variety of information can be selected and compared from different tables. Calculations can be made on the data at the data base level, allowing for fewer intermediate programs to be necessary for formatting or scaling of data.

The relational data base we used on the microcomputer was Oracle. Oracle offers several advantages to a system. First, Oracle is available for mainframe computer, minicomputer, and microcomputer use. This means that a user who may have access to both a minicomputer and a microcomputer needs to learn only one data base system. The sharing of data and data extraction is also more attractive when the data base systems are the same. Oracle's command language is easy to learn by anyone who knows a programming language. In fact, Oracle's command language, SQL, is non-procedural so it is more easily learned than BASIC. The user must only understand the basic constructs of SQL and the logical implementation of those constructs.

Documentation utilities assist the user in the development of the data dictionary for the particular data base design. With them the user can obtain the structure of the tables he has produced, a listing of the various tables and indices he has developed (by name, structure type, and filetype), data elements (by name, type, source, and definition), and a listing of the data structures showing the particular elements that they contain. Oracle only provides some of these. For the sake of clarity we created the tables to produce these definitions for our system in the microcomputer implementation.

User control and security are easily implemented through Oracle. Users can create views and determine the types of access they wish to assign to other users of

their data. Additionally, the data base administrator or the user acting as the data base administrator can determine the types of access all users will have to the data base.

2. **Longterm Data Base Considerations**

With the need of a historical data base clearly identified, the use of the microcomputer alone does appear to be feasible. The amount of data and overhead of a DBMS on a microcomputer would be prohibitive. Therefore, an extraction system is necessary for the minicomputer. Once a data base is established on the minicomputer, the microcomputer can extract the data for a particular cost center as needed or a user could use a system implemented on the minicomputer.

VII. CONCLUSIONS AND RECOMMENDATIONS

A. CONCLUSIONS

The methodology we followed was a task organized approach. We felt that no one approach of the major authorities on DSS development was complete in and of itself. The task organized approach allowed for structured documentation tools to be used with an iterative approach.

The iterative approach was central to our methodology. Design and implementation must be accomplished in quick, as short as possible cycles, in this iterative approach. This allows the developer to interact with the user, to obtain rapid feedback on the progress and the direction of the project. Communication is very important to a DSS project.

Often the problem definition in a DSS cannot be as precise as in a MIS project. The semistructured or unstructured nature of DSS projects can make a clear cut problem definition impossible. By approaching a problem that is not very well defined in quick, rapid, iterative steps, the problem will become clearer as the iterations progress.

The proper environment must be established before a DSS can be initiated. When initiating a DSS, minimization of risk is crucial. If an initial project is not successful, it may be a long time before subsequent attempts at a DSS are made, even if a DSS is needed.

In order to minimize risks, a champion or strong enthusiast must be found. This is a person who can envision the benefits of a DSS. If there is no proponent of the system, or it is forced on someone, the iterative approach will not be effective. User relations is of paramount importance and must be established immediately. Being prepared and not bringing any surprises is the best way to keep your relations in good standing.

Once the enthusiast is found, a decision must be focused upon. The DSS should focus on one decision or one problem. This limits the scope and allows the developer to narrow in on one area. A user, upon hearing the term DSS, may get visions of a system that will answer his every question within seconds. By focusing on one decision, the developer can guide the user through the project without building false hopes or misleading expectations.

Within our methodology, we utilized structured techniques during the analysis and design. These techniques included the use of data flow diagrams, structure charts, and a data dictionary. We feel that these are among the best documentation tools available today. The use of these tools reduce maintenance costs later. Further development is also made easier with the structured documentation. This is especially true when the original developers are not going to be involved in any further development.

When the enthusiast is found, the decision is identified and the structured tools of analysis and design are ready, Sprague and Carlson's ROMC approach further focuses the effort [Ref. 2]. Representations let the developer focus on the displays the user desires, the logical design. With the representations, the data and data base must be investigated and documented. The data base is the keystone of the DSS. Operations should follow easily with the representations. The manipulation of data and displays are the required operations. Memory aids remind the user of what needs to be done or what can be done. The controls for the system can then be designed to interface the parts into a functioning, useful system.

During our iterative steps, we began a similar development on a completely different system. Although this is not recommended for every DSS development, it helped us focus on problems that we had been overlooking in the initial prototype. By using the same design documented with structured techniques, we were able to quickly develop the similar system on a microcomputer. This gave us the opportunity to compare the systems. This comparison helped identify some problems on both systems that may not have been identified until later in the project.

B. RECOMMENDATIONS

Mare Island Naval Shipyard has several alternatives available to them at this point. We recommend that the data base for the minicomputer be implemented. This will allow a better comparison between the minicomputer and the microcomputer. Since the documentation is available from the microcomputer implementation and the design of the data base is complete and can be used with any relational or data base that is logically relational, this step should encompass only 20 manhours, fewer if the programmer is familiar with the tools. Next, or even concurrently, the extraction of the data from SABRS must be completed. The extraction of the data is not a trivial problem, although technically it is feasible. The extraction is necessary to get the data

into the data base whether it is on the minicomputer or the microcomputer. We estimate the data extraction from SABRS into a data base to be a 160 manhour job. Our estimate is based on our work with this project and not on any previous data extraction experience.

With the data base and data extraction, data integrity and accessibility will have to be defined. Since an historical data base is desired, the data base should not be able to be changed by the normal user. This can be accomplished easily in a data base system such as Oracle.

Once the data base is established and the extraction system is in place, several alternatives must be decided upon. If the system does not appear to be feasible, the project can be stopped. If the project continues, additional cost centers can be added iteratively. This would be a safe course to follow. In that case, the control programs would have to be modified to include the new cost centers, and the graphics modules for each new graph would need to be written. These would be the modules that are appended onto the main module for each graph.

The feasibility of tying CCA in with SABRS is another possibility. SABRS offers the user a "what if" capability. If SABRS could be integrated with the graphics of CCA, a more powerful DSS would result at a minimal cost. The difficulty and compatibility of the systems should be determined when the extraction of the data is accomplished.

Whether the minicomputer implementation or the microcomputer implementation is better is another question that must be answered. A microcomputer should not be the repository for a large historical data base. However, a microcomputer could be used as the workspace for the decision maker. This would allow the decision maker the opportunity to change his data as he wishes without affecting the centralized data repository. Microcomputers allow the user to view the graphics at his desk, where the minicomputer CCA would require a graphics terminal or a hard copy to be created for the decision maker.

The minicomputer should be the repository for an historical data base. This will simplify the extraction problem, from minicomputer to minicomputer. The graphics modules on the minicomputer are easier to create and maintain, and a sophisticated user can learn TEL-A-GRAF at the command level, creating his own graphics. This is much more difficult on the microcomputer. The graphics on the microcomputer are written in C and are not written in a command language style like TEL-A-GRAF.

C. SUMMARY

The best way to approach a DSS problem is with a DSS methodology. An iterative approach that uses the tools of structured analysis and design provides the developer with the best of both worlds. The necessary documentation for a project is completed and the short cycles of the iterative approach help to promote communications.

Mare Island Naval Shipyard should complete the data base implementation and the data extraction system. Expanding that system to include all the cost centers, iteratively, one at a time, is a minimal risk alternative. Interfacing CCA directly with SABRS would be another inexpensive alternative. Whether the networked minicomputers or the microcomputers are most desirable to a decision maker is still questionable. Perhaps it should be left to the individual decision maker to decide.

Further research is indicated in the area of data extraction. The usefulness of the microcomputers in the offices in five years is another research question that can be investigated. Finally, the direction that the DSS should take after the successful implementation of the CCA is an important area for follow-on research. This would include an overall design of a DSS for the Shipyard.

APPENDIX A
STRUCTURED SPECIFICATION

1. DATA FLOW DIAGRAM

Data flow diagrams show the "flow of data, not of control." The symbols used are [Ref. 4: p. 40]:

1. The named vector (called a data flow), which portrays a data path.
2. The bubble (called a process), which portrays transformation of data.
3. (Two parallel straight lines) which portray a file or data base.
4. The box (called a source or sink), which portrays a net originator or receiver of data - typically a person or an organization outside the domain of our study.

"The Data Flow Diagram is documentation of a situation from the point of view of the data." [Ref. 4: p. 41] It will provide the user a clear understanding of the present situation, and also the data required for the operation of the system. In addition, any errors in the system description can be more easily identified by other analysts.

The Data Flow Diagram is developed hierarchically. In this case the Top Level diagram is in Table 4, the First Level diagram is in Table 5, and the Second Level diagram of Process 1.0, of the First Level, is in Table 6

2. MINISPECIFICATION

We were not concerned with the exact details of how the users accomplished the processes in the Data Flow Diagram. The primary consideration was to get a general idea of what was going on, model it, and attempt to develop a system that would assist them.

1.1 Select Data

1. Gather Budget Input Reports provided by the Comptroller Department.
2. Select the data to be used for the desired report.

1.2 Insertion of data

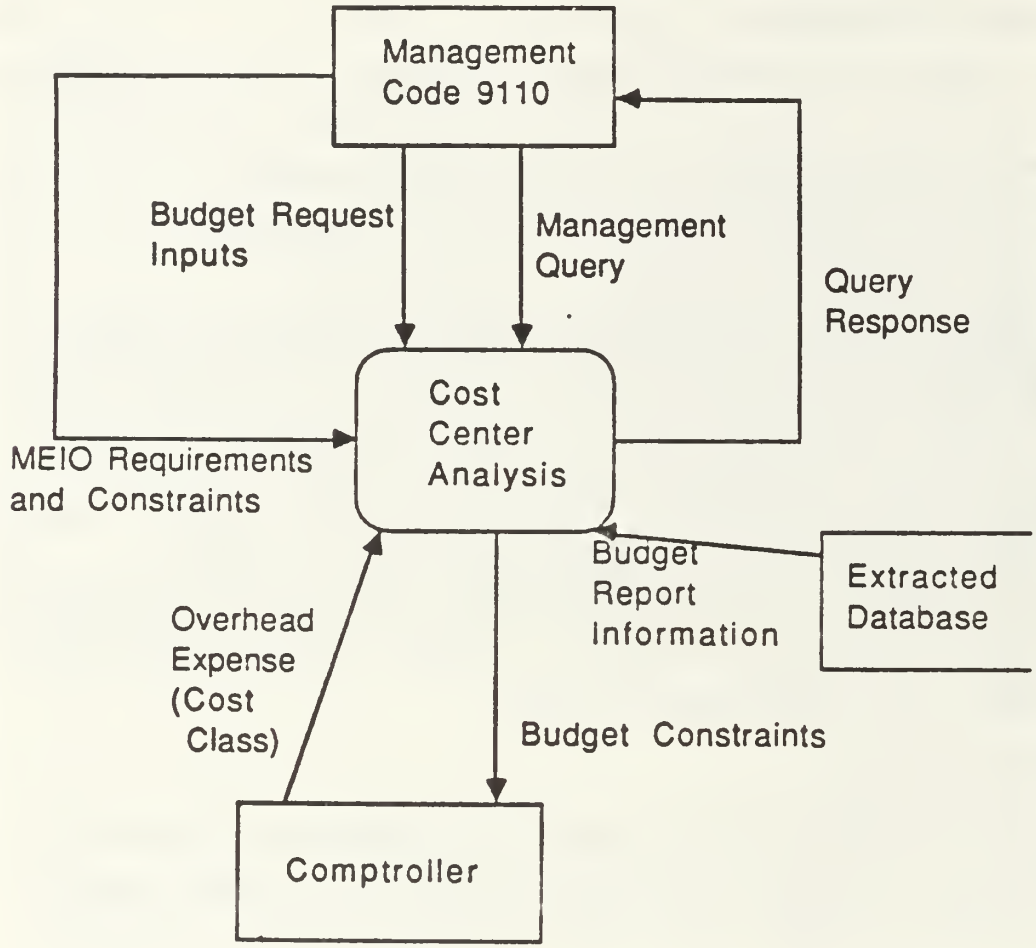
1. Input the selected data to a data file.
2. The extracted data is used to prepare the desired report.

2.0 Prepare Reports

1. Input the management query
2. Identify the type of report that will answer the query.
3. Determine if:

TABLE 4

TOP LEVEL DATA FLOW DIAGRAM



* Extracted data from budget input reports will be needed, or
* Overhead expenses (cost class), or
* Actual to budget comparison will support the report selected.

4. Manipulate and format the input data as requested in the management query.
5. Provide query response to management.

3.0 Prepare Budget

1. Identify Comptroller budget constraints
2. Identify MEIO requirements and constraints.

TABLE 5

FIRST LEVEL DATA FLOW DIAGRAM

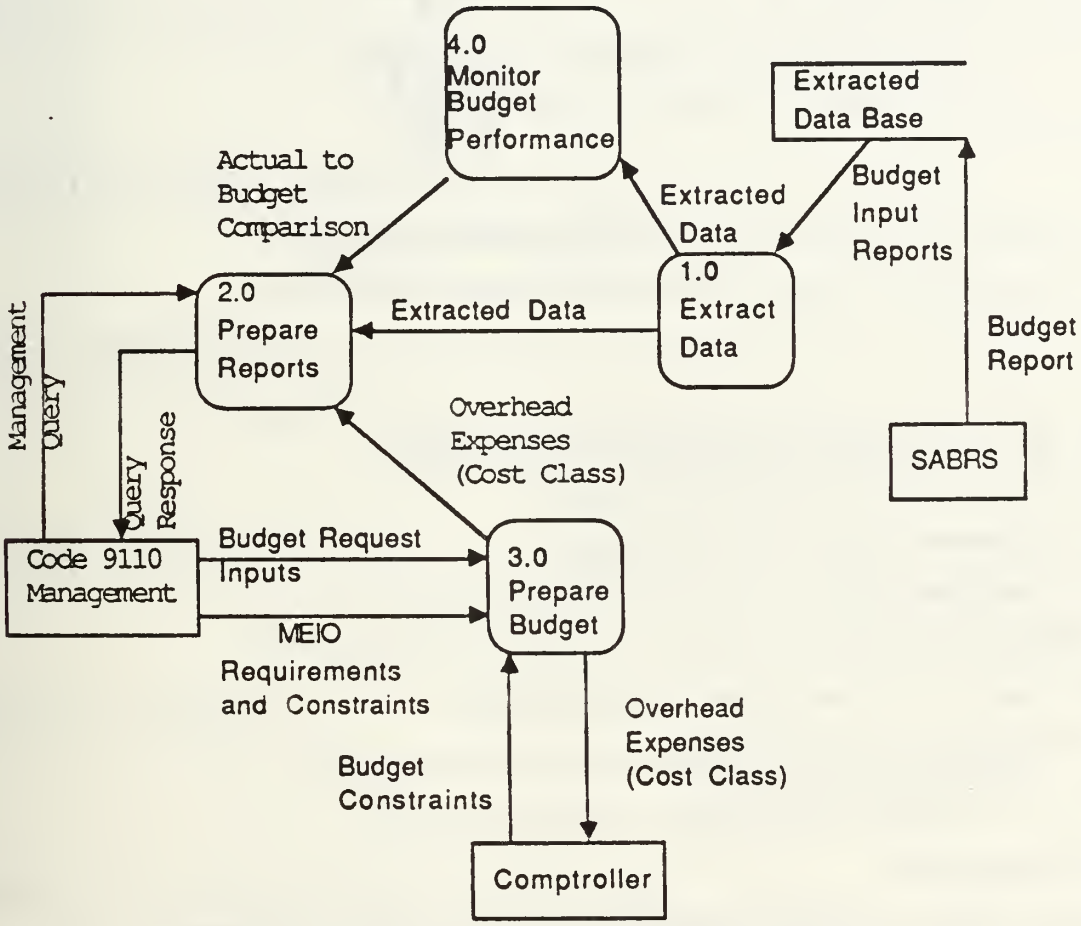
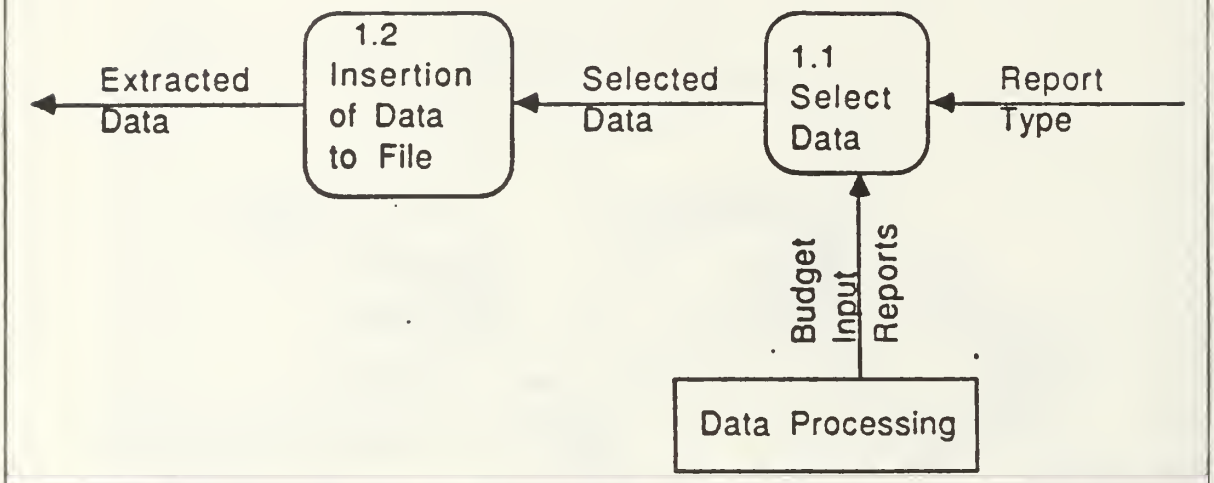


TABLE 6

SECOND LEVEL DATA FLOW DIAGRAM OF PROCESS 1.0



3. Request and identify budget request inputs within MEIO.
4. Develop MEIO overhead expenses (cost class).
5. Obtain management approval for the budget.
6. Submit to the Comptroller.

4.0 Monitor Budget Performance

1. Analyze extracted data from SBR-22A and SBR-22B.
2. Determine progress and errors, if any.
3. Report to management if requested.

3. DATA DICTIONARY

This section provides a rigorous description of the data that is depicted in the Data Flow Diagram. Before the completion of this section we had only a cursory understanding of the data involved. The disciplined analysis, which involved breaking larger data flows into data elements, brought a great deal of clarity to our understanding of the data.

The Data Dictionary is organized alphabetically to assist the reader in locating particular documents or data elements.

ACQUISITION OF MINOR PROPERTY (68) =

* COST OF PURCHASED OR
MANUFACTURED MINOR PROPERTY,
WHICH IS DEFINED AS THOSE COSTING
LESS THAN $1000 *

ACTUAL MATERIAL AMOUNT =
* TOTAL DOLLAR AMOUNT OF MATERIAL COSTS
INCURRED TO DATE WITHIN A COST FUNCTION *

ACTUAL OT HOURS =
* OVERTIME MANHOURS CHARGED TO
DATE WITHIN A COST FUNCTION *

ACTUAL OT LABOR AMOUNT =
* TOTAL DOLLAR AMOUNT OF OVERTIME LABOR
CHARGED TO DATE WITHIN A COST FUNCTION *

ACTUAL OT M/P/D =
* OVERTIME MAN PER DAY ACTUALLY INCURRED TO
DATE WITHIN A COST FUNCTION *

ACTUAL OTHER =
* TOTAL DOLLAR AMOUNT OF OTHER
INCURRED TO DATE WITHIN A COST FUNCTION *

ACTUAL ST HOURS =
* STRAIGHT TIME MANHOURS CHARGED TO
DATE WITHIN A COST FUNCTION *

ACTUAL ST LABOR AMOUNT =
* TOTAL DOLLAR AMOUNT OF STRAIGHT TIME LABOR
CHARGED TO DATE WITHIN A COST FUNCTION *

ACTUAL ST M/P/D =
* STRAIGHT MAN PER DAY ACTUALLY INCURRED TO
DATE WITHIN A COST FUNCTION *

ACTUAL TOTAL AMOUNT =
* TOTAL DOLLAR AMOUNT INCURRED FOR A COST

FUNCTION *

ADMINISTRATION (9112) =
* ALL LABOR AND OTHER COSTS IDENTIFIABLE TO THE ADMINISTRATION OF THE DATA PROCESSING OFFICE AND OVERHEAD COSTS NOT ASSIGNABLE TO OTHER FUNCTIONAL SUBDIVISIONS *

AVE OT RATE =
* AVERAGE HOURLY RATE FOR OVERTIME (BASED ON INDIVIDUAL RATES) *

AVERAGE BASE =
* AVERAGE HOURLY RATE ACCELERATED BY 32-1/2% TO ACCOUNT FOR BENEFITS AND LEAVE *

BUDGET CONSTRAINTS =
* CONSTRAINTS SET BY THE SHIPYARD COMPTROLLER (IE. ANNUAL LEAVE WILL NOT EXCEED 14%) *

BUDGET REQUEST INPUTS =
* MANAGEMENT REQUESTS FOR INCLUSION OF PARTICULAR ITEMS WITHIN THE DEPARTMENTAL BUDGET *

BUDGET VS ACTUAL PERFORMANCE REPORT (SBR-22A) =
ISSUE DATE + DATA DATE +
{COST CENTER} + {COST FUNCTION}
+ {COST CLASS} + {ST HOURS} +
{OT HOURS} + {ST M/P/D} + {OT
M/P/D} + {ST LABOR} + {OT LABOR}
+ {MATERIAL AMOUNT} + {OTHER AMOUNT}
+ {TOTAL AMOUNT}]

BUDGETED MATERIAL AMOUNT =
* TOTAL DOLLAR AMOUNT OF MATERIAL BUDGETED TO BE INCURRED FOR THE FISCAL YEAR WITHIN A COST FUNCTION *

BUDGETED OT HOURS =

* OVERTIME MANHOURS BUDGETED FOR THE FISCAL YEAR WITHIN A COST FUNCTION *

BUDGETED OT LABOR AMOUNT =
* TOTAL DOLLAR AMOUNT OF OVERTIME LABOR BUDGETED FOR THE FISCAL YEAR TO BE INCURRED WITHIN A COST FUNCTION *

BUDGETED OT M/P/D =
* OVERTIME MAN PER DAY BUDGETED FOR THE FISCAL YEAR WITHIN A COST FUNCTION *

BUDGETED OTHER =
* TOTAL DOLLAR AMOUNT OF OTHER BUDGETED TO BE INCURRED FOR THE FISCAL YEAR WITHIN A COST FUNCTION *

BUDGETED ST HOURS =
* STRAIGHT TIME MANHOURS BUDGETED FOR THE FISCAL YEAR WITHIN A COST FUNCTION *

BUDGETED ST LABOR AMOUNT =
* TOTAL DOLLAR AMOUNT OF STRAIGHT TIME LABOR BUDGETED FOR THE FISCAL YEAR TO BE INCURRED WITHIN A COST FUNCTION *

BUDGETED ST M/P/D =
* STRAIGHT MAN PER DAY BUDGETED FOR THE FISCAL YEAR WITHIN A COST FUNCTION *

BUDGETED TOTAL AMOUNT =
* TOTAL DOLLAR AMOUNT BUDGETED FOR THE FISCAL YEAR FOR A COST FUNCTION *

CATEGORY TITLE = * FUNCTION COST CLASS *

CONSUMEABLE SUPPLIES (12) =
* MATERIAL COSTS OF CONSUMEABLE, REUSABLE, AND MINOR NON-CONSUMEABLE

SUPPLIES AND MATERIALS NOT
OTHERWISE CHARGEABLE TO ANOTHER
COST CLASS, OR AS DIRECT MATERIAL
TO PRODUCTIVE JOB ORDERS *

CONSUMEABLE SUPPLIES AND INSTALLATION (97) =
* COST OF CONSUMEABLE SUPPLIES RELATED TO
THE ADP FUNCTION; ALSO CHARGED WITH IN-
HOUSE COSTS ASSOCIATED WITH THE
INSTALLATION OF ADP MINOR PROPERTY *

CONTRACTUAL SERVICES (96) =
* COSTS OF CONTRACTUAL SERVICES (FOR
EXAMPLE, TIME SHARING OR DATA ENTRY
SUPPORT) OTHER THAN THOSE SERVICES
SPECIFIED AS CHARGED TO COST CLASSES
94 AND 95 *

CONTROL AND SCHEDULING (9116) =
* ALL LABOR AND OTHER COSTS
IDENTIFIABLE AS OVERHEAD OF THE
CONTROL AND SCHEDULING
FUNCTION, EXCEPT FOR COSTS
IDENTIFIED TO COST FUNCTION
9119 *

COST CENTER = COST CENTER NUMBER + COST CENTER NAME

COST CENTER NAME =
[DATA PROCESSING OFFICE/MANAGEMENT
ENGINEERING OFFICE]

COST CENTER NUMBER = [9|1|1/4 0]

COST CLASS (NUMBER) =
[SUPERVISION GRADED (02)| NON-SUPERVISION
GRADED (03)| SHOP GENERAL (04)| MATERIAL
(04)| CONSUMEABLE SUPPLIES (12) UNALLOCATED

(19)| TRAVEL (30)|
DUPLICATING/MICROFICHE/ILLUSTRATORS (33)
TRAINING (39) ACQUISITION OF MINOR PROPERTY
(68)| SUPERVISION GRADED (91)| NON-
SUPERVISION, ANALYSIS AND PROGRAMMING (92)|
NON-SUPERVISION GRADED, OTHERS (93) RENTAL
AND COMMUNICATION (94)| MAINTENANCE (95)|
CONTRACTUAL SERVICES (96) CONSUMEABLE SUPPLIES
AND INSTALLATION (97)| MINOR PROPERTY (98)|
TRAINING (99)]

COST CLASS NO. = * COST CLASS NUMBER *

COST FUNCTION =
[MIS IMPROVEMENT ADP PROGRAMS (9111){
ADMINISTRATION (9112){ PROGRAMMING (9113){
RENT OF EQUIPMENT AND INSTALLATION COST (9114){
OPERATIONS (9115){ CONTROL AND SCHEDULING (9116){
EDP OPERATIONS (9117)| EAM OPERATIONS (9118)|
NAVSHIPS NSY MIS PROGRAM (9119){ MANAGEMENT
ENGINEERING OFFICE ADMINISTRATION (9142){
MANAGEMENT SYSTEM SUPPORT (9143){
QC/PRODUCTIVITY (9144){]

DATA DATE = * EFFECTIVE DATE OF DATA USED FOR THE REPORT *

DEPARTMENTAL SUMMARY BY COST CLASS AND SHIPYARD TOTAL (BUDGET VS ACTUAL)=
ISSUE DATE + DATA DATE +
{COST CENTER} + {COST CLASS} +
{ST HOURS} + {OT HOURS} + {ST M/P/D}
+ {OT M/P/D} + {ST LABOR} + {OT LABOR}
+ {MATERIAL AMOUNT} + {OTHER AMOUNT}
+ {TOTAL AMOUNT}]

DUPLICATING/MICROFICHE/ILLUSTRATORS (33) =
* COST OF ALL PURCHASED

PRINTING, REPRODUCTION AND
DUPLICATING WHEN NOT CHARGEABLE
TO A PARTICULAR CUSTOMER ORDER *

EAM OPERATIONS (9118) =
* ALL LABOR AND OTHER COSTS IDENTIFIABLE
AS OVERHEAD OF THE EAM OPERATIONS
FUNCTION, EXCEPT FOR COSTS IDENTIFIED TO
COST FUNCTION 9119 *

EDP OPERATIONS (9117) =
* ALL LABOR AND OTHER COSTS IDENTIFIABLE
AS OVERHEAD OF THE EDP OPERATIONS
FUNCTION, EXCEPT FOR COSTS IDENTIFIED TO
COST FUNCTION 9119 *

FUNDS ADMIN (CODE NO.) =
* CODE OF FUNDS ADMINISTRATOR FOR
MANAGEMENT ENGINEERING AND
INFORMATION OFFICE (014/016) *

INPUT DATA =
[COST CENTER/FUNCTION BUDGET VS ACTUAL
PERFORMANCE REPORT (SBR-22A)| DEPARTMENTAL
SUMMARY BY COST CLASS AND SHIPYARD TOTAL
(BUDGET VS ACTUAL) (SBR-22B)]

ISSUE DATE = * DATE REPORT WAS SUBMITTED TO USER *

MAINTENANCE (95) = * MAINTENANCE COSTS OF ADP EQUIPMENT *

MANAGEMENT ENGINEERING OFFICE (9142) =
* ALL LABOR AND OTHER
COSTS IDENTIFIED AS OVERHEAD OF
THE DIRECTOR OF MANAGEMENT
ENGINEERING OFFICE, AND OTHER
COSTS WHICH ARE NOT ASSIGNABLE TO

ANOTHER FUNCTION OF THE MEO *

MANAGEMENT SYS SUPPORT (9143) =
* ALL LABOR AND OTHER COSTS FOR PERFORMING THE MEO FUNCTION *

MANAGER QUERY =
* AD HOC QUERIES CONCERNING BUDGET PREPARATION, CONTROL AND VARIANCE ANALYSIS *

MATERIAL (04) = * SYNONYM FOR SHOP GENERAL *

MATERIAL AMOUNT =
[BUDGETED MATERIAL AMOUNT + ACTUAL MATERIAL AMOUNT + PERCENTAGE OF MATERIAL AMOUNT + MATERIAL VARIANCE]

MATERIAL VARIANCE =
* ACTUAL OT LABOR MINUS THE AMOUNT OF THE BUDGETED OT LABOR THAT SHOULD HAVE BEEN EXPENDED TO DATE (ACTUAL MINUS THE PRODUCT OF THE PERCENT OF THE PERIOD ELAPSED AND BUDGET) *

MEIO REQUIREMENTS AND CONSTRAINTS =
* CONSTRAINTS SET BY THE MEIO MANAGEMENT *

MEN GROSS = * TOTAL NUMBER OF PERSONNEL *

MEN IVB =
* COMPUTED TOTAL NUMBER OF PERSONNEL MINUS THOSE ON LEAVE *

MEN OTHER = * PERSONNEL BORROWED BETWEEN COST CENTERS *

MEN TOTAL = * THE TOTAL OF MEN IVB AND MEN OTHER *

MINOR PROPERTY (98) =
* PURCHASED COSTS OF ADP MINOR PROPERTY, WHEN THAT COST IS LESS THAN $1000 OR THE ITEM HAS

A USEFUL LIFE OF TWO YEARS OR LESS REGARDLESS OF COST *

NAVSHIPS NSY MIS PROGRAM (9119) =
* ALL LABOR AND OTHER COSTS IDENTIFIABLE TO DEVELOPMENT AND MAINTENANCE OF NAVSEA NAVSHIPYD MIS ASSIGNMENTS *

NON-SUPERVISION GRADED (03) =
* INDIRECT LABOR COST OF NON-SUPERVISORY GRADED PERSONNEL *

NON-SUPERVISION, ANALYSIS AND PROGRAMMING (92) =
* LABOR COSTS OF PERSONNEL WHILE ENGAGED IN ADP ANALYSIS AND PROGRAMMING *

NON-SUPERVISION GRADED, OPERATIONS (9115) =
* ALL LABOR AND OTHER COSTS IDENTIFIABLE AS OVERHEAD FOR SUPERVISING AND ADMINISTERING THE OPERATIONS DIVISION, EXCEPT FOR COST IDENTIFIED TO COST FUNCTION 9119 *

OT HOURS =
[BUDGETED OT HOURS + ACTUAL OT HOURS + PERCENTAGE OF OT HOURS + OT HOURS VARIANCE]

OT HOURS VARIANCE =
* ACTUAL OT HOURS MINUS THE AMOUNT OF THE BUDGETED OT HOURS THAT SHOULD HAVE BEEN EXPENDED TO DATE (ACTUAL MINUS THE PRODUCT OF THE PERCENT OF THE PERIOD ELAPSED AND BUDGET) *

OT LABOR AMOUNT =
[BUDGETED OT LABOR AMOUNT + ACTUAL OT LABOR AMOUNT + PERCENTAGE OF OT LABOR AMOUNT + OT LABOR VARIANCE]

OT LABOR VARIANCE =

* ACTUAL OT LABOR MINUS THE AMOUNT OF THE BUDGETED OT LABOR THAT SHOULD HAVE BEEN EXPENDED TO DATE (ACTUAL MINUS THE PRODUCT OF THE PERCENT OF THE PERIOD ELAPSED AND BUDGET)

OT M/P/D =

[BUDGETED OT MAN PER DAY + ACTUAL OT MAN PER DAY + PERCENTAGE OF OT M/P/D + OT M/P/D VARIANCE]

OT M/P/D VARIANCE =

* ACTUAL OT M/P/D MINUS THE AMOUNT OF THE BUDGETED OT M/P/D THAT SHOULD HAVE BEEN EXPENDED TO DATE (ACTUAL MINUS THE PRODUCT OF THE PERCENT OF THE PERIOD ELAPSED AND BUDGET)

OTHER COSTS =

* OTHER BUDGETED COSTS INCLUDING PRIMARILY CONTRACTS AND TRAVEL *

OTHER (PRIMARILY CONTRACTS AND TRAVEL) =

[BUDGETED OTHER + ACTUAL OTHER + PERCENTAGE OF OTHER + OTHER VARIANCE]

OTHER VARIANCE =

* ACTUAL OTHER MINUS THE AMOUNT OF THE BUDGETED OTHER THAT SHOULD HAVE BEEN EXPENDED TO DATE (ACTUAL MINUS THE PRODUCT OF THE PERCENT OF THE PERIOD ELAPSED AND BUDGET)

OTHERS (OPERATIONS) (93) =

* LABOR COSTS OF PERSONNEL (OTHER THAN THOSE SPECIFIED AS CHARGED TO COST CLASS 91 AND 92) WHOSE PRINCIPAL DUTIES ARE DIRECTLY RELATED TO CONDUCTING OR SUPPORTING

THE ADP FUNCTION *

OVERHEAD EXPENSES BY COST CLASS =
[FUNDS ADMIN (CODE NO.) + COST CENTER NO. + STRAIGHT TIME WORKING HOURS + {COST CLASS NO.} + {CATEGORY TITLE} + {MEN GROSS} + {MEN IVB} + {MEN OTHER} + {MEN TOTAL} + {STRAIGHT HOURS} + {AVERAGE BASE} + {STRAIGHT LABOR SS} + {OVT MEN} + {OVT HOURS} + {AVE OT RATE} + {OVT LABOR} + {TOTAL MNDAY} + TOTAL HOURS} + {TOTAL LABOR SS} + TOTAL MATER} + {OTHER COSTS} + {TOTAL COST} + {TOT EXPEN}]

OVT HOURS = * NUMBER OF OVERTIME HOURS *

OVT LABOR =
* PRODUCT OF OVERTIME HOURS AND AVERAGE OVERTIME RATE *

OVT MEN = * PERSONNEL ON OVERTIME (NOT USED) *

PERCENTAGE OF MATERIAL AMOUNT =
* PERCENTAGE OF BUDGETED MATERIAL AMOUNT INCURRED TO DATE WITHIN A COST FUNCTION *

PERCENTAGE OF OT HOURS =
* PERCENTAGE OF BUDGETED TOTAL OVERTIME TIME MANHOURS ACTUALLY INCURRED TO DATE WITHIN A COST FUNCTION *

PERCENTAGE OF OT LABOR AMOUNT =
* PERCENTAGE OF BUDGETED OVERTIME LABOR AMOUNT INCURRED TO DATE *

PERCENTAGE OF OTHER =
* PERCENTAGE OF BUDGETED OTHER

AMOUNT INCURRED TO DATE WITHIN A
COST FUNCTION *

PERCENTAGE OF ST HOURS =
* PERCENTAGE OF BUDGETED TOTAL STRAIGHT
TIME MANHOURS ACTUALLY INCURRED TO
DATE WITHIN A COST FUNCTION *

PERCENTAGE OF ST LABOR AMOUNT =
* PERCENTAGE OF BUDGETED STRAIGHT TIME LABOR
AMOUNT INCURRED TO DATE *

PERCENTAGE OF TOTAL AMOUNT =
* PERCENTAGE OF BUDGETED TOTAL
DOLLAR AMOUNT INCURRED TO DATE
WITHIN A COST FUNCTION *

PROGRAMMING (9113) =
* ALL LABOR AND OTHER COSTS IDENTIFIABLE AS
OVERHEAD OF THE ANALYSIS AND PROGRAMMING
DIVISION, EXCEPT FOR COSTS IDENTIFIED TO
COST FUNCTION 9119 *

QC/PRODUCTIVITY (9144) =
* ALL LABOR AND OTHER COSTS ASSOCIATED
WITH THE PRODUCTIVITY IMPROVEMENT PROGRAM *

QUERY RESPONSE =
* BUDGET ANALYST RESPONSE TO AD HOC QUERIES*

RENTAL AND COMMUNICATION (94) =
* ALL ADP EQUIPMENT RENTALS,
INCLUDING RELATED ANCILLARY
COMMUNICATION EQUIPMENT RENTALS;
ALSO CHARGED WITH TELEPHONE
COMMUNICATION SERVICE COSTS
ASSOCIATED WITH UNIQUE OR
DEDICATED LINES USED IN SUPPORT

OF THIS EQUIPMENT *

RENT OF EQUIPMENT AND INSTALLATION COST (9114) =
* ALL COSTS OF ADP/EAM RENTAL AND MAINTENANCE INCLUDING THE COST OF RENTING TERMINALS EXCLUDING MINICOMPUTERS CHARGEABLE TO BENEFITING COST CENTERS AND COST CLASS 37 *

SHOP GENERAL (04) =
* INDIRECT COSTS OF SUPPLIES AND LABOR OF NON-SUPERVISORY UNGRADED PERSONNEL WHILE ENGAGED IN WORK OR A GENERAL OVERHEAD NATURE BUT NOT OTHERWISE CHARGEABLE TO ANOTHER COST CLASS OR AS DIRECT LABOR *

ST HOURS =
[BUDGETED ST HOURS + ACTUAL ST HOURS + PERCENTAGE OF ST HOURS + ST HOURS VARIANCE]

ST HOURS VARIANCE =
* ACTUAL ST HOURS MINUS THE AMOUNT OF THE BUDGETED ST HOURS THAT SHOULD HAVE BEEN EXPENDED TO DATE (ACTUAL MINUS THE PRODUCT OF THE PERCENT OF THE PERIOD ELAPSED AND BUDGET) *

ST LABOR AMOUNT =
[BUDGETED ST LABOR AMOUNT + ACTUAL ST LABOR AMOUNT + PERCENTAGE OF ST LABOR AMOUNT + ST LABOR VARIANCE]

ST LABOR VARIANCE =
* ACTUAL ST LABOR MINUS THE AMOUNT OF THE BUDGETED ST LABOR THAT SHOULD HAVE BEEN EXPENDED TO DATE (ACTUAL MINUS THE PRODUCT OF THE PERCENT OF THE PERIOD ELAPSED AND BUDGET) *

ST M/P/D =

[BUDGETED ST MAN PER DAY + ACTUAL ST MAN PER DAY + PERCENTAGE OF ST M/P/D + ST M/P/D VARIANCE]

ST M/P/D VARIANCE =

* ACTUAL ST M/P/D MINUS THE AMOUNT OF THE BUDGETED ST M/P/D THAT SHOULD HAVE BEEN EXPENDED TO DATE (ACTUAL MINUS THE PRODUCT OF THE PERCENT OF THE PERIOD ELAPSED AND BUDGET)

STRAIGHT HOURS = * SYNONYM FOR STRAIGHT TIME WORKING HOURS *

STRAIGHT LABOR SS =

* PRODUCT OF AVERAGE BASE AND STRAIGHT HOURS *

STRAIGHT TIME WORKING HOURS =

* LABOR HOURS WITH LEAVE SUBTRACTED (NUMBER OF HOURS * 2008 HOURS AVAILABLE IN A YEAR) *

SUPERVISION GRADED (02) =

* INDIRECT LABOR COST OF GRADED SUPERVISORY PERSONNEL WHILE ENGAGED IN THE SUPERVISION OF OTHERS *

SUPERVISION GRADED (91) =

* LABOR COST OF PERSONNEL WHILE ENGAGED IN THE SUPERVISION AND DIRECTION OF PERSONNEL PERFORMING ADP FUNCTIONS *

TOTAL AMOUNT =

[BUDGETED TOTAL AMOUNT + ACTUAL TOTAL AMOUNT + PERCENTAGE OF TOTAL AMOUNT + TOTAL VARIANCE]

TOTAL COSTS =

* TOTAL BUDGETED COSTS (SUM OF TOTAL LABOR, TOTAL

MATER, AND OTHER COSTS *

TOTAL HOURS = *SUM OF ST HOURS AND OT HOURS *

TOTAL LABOR SS =
* DOLLAR SUM OF THE PRODUCT OF ST HOURS AND AVE BASE AND THE PRODUCT OF OT HOURS AND AVE OT RATE *

TOTAL MATER = * TOTAL BUDGETED MATERIAL COST *

TOTAL MNDAY = * TOTAL PERSONNEL (OVERTIME AND MEN) *

TOTAL VARIANCE =
* ACTUAL TOTAL MINUS THE AMOUNT OF THE BUDGETED TOTAL THAT SHOULD HAVE BEEN EXPENDED TO DATE (ACTUAL MINUS THE PRODUCT OF THE PERCENT OF THE PERIOD ELAPSED AND BUDGET)

TRAINING (39) =
* INDIRECT EXPENSES INCIDENT TO ORGANIZED TRAINING PROGRAMS EXCEPT APPRENTICE AND NUCLEAR TRAINING PROGRAMS *

TRAINING (99) =
* TRAINING COSTS IN SUPPORT OF THE ADP FUNCTION *

TRAVEL (30) =
* COST OF APPROVED TRAVEL, INCLUDING SUBSISTENCE, WHEN NOT CHARGEABLE TO A PARTICULAR CUSTOMER ORDER OR COST CLASS BY TYPE *

UNALLOCATED (CODING REJECTS) (19) =
* COSTS WHICH CANNOT BE IDENTIFIED WITH A CUSTOMER ORDER OR AN ESTABLISHED EXPENSE ACCOUNT *

4. AUTOMATED DATA DICTIONARY

a. Method

The data dictionary is automated, although it is not an integral part of the Cost Center Analysis system. Instead of being a part of the system, it explains the various parts of the the system. This data dictionary was set in the form of data tables in Oracle. Although some of this information can be generated from Oracle system utilities, it was felt that more specific information was necessary for this system's documentation.

b. Data Representation

The following tables represent two information tables. These are representative of the structure shown in the system data structure diagrams (Bachman diagrams). Although this system was designed for personnel who are intimately familiar with the Cost Center terminology, we attempted to explain as much as possible data meanings and abbreviations. This was done to facilitate the work of follow-on designers and implementors.

In addition, there are five systems description tables which provide the data base structure. These make up a ready reference for follow-on implementors, reducing the chances of misunderstanding.

c. Data Maintenance

This data dictionary has been designed for an extracted data base. It would be kept current by biweekly downloads from the shipyard's Prime Network. As such the system would grow to considerable size by the end of the fiscal year.

These system updates would not change the structure of the data dictionary as depicted. It would only change the number of records within the tables.

d. Data Security

Physical security controls are adequate for this system. Physical internal controls will ensure the integrity of the data base. These controls are already in place within the office workspaces.

If information security became more critical, these controls could be implemented through the Oracle DBMS, by using passwords and controlling access.

e. Back-up and Recovery

This is provided by the Oracle DBMS as part of its services to the users. Oracle uses a before image file to provide the recovery information.

f. **Budget Table Structure**

```
UFI> SELECT * FROM COL WHERE TNAME = 'BUDGET'

TNAME                          COLNO  CNAME
                               COLTYP WIDTH
  SCALE NULLS

BUDGET                             1  COST_FUN_NO
                               CHAR      4
      NOT NULL

BUDGET                             2  COST_CL_NO
                               CHAR      2
      NOT NULL

BUDGET                             3  ST       HOURS
                               NUMBER        7
    1 NULL

BUDGET                             4  OT       HOURS
                               NUMBER        7
    1 NULL

BUDGET                             5  ST       LABOR
                               NUMBER        7
    1 NULL

BUDGET                             6  OT       LABOR
                               NUMBER       11
    4 NULL

BUDGET                             7  MATERIAL
                               NUMBER       11
    4 NULL
```

```
BUDGET                          8                       OTHER
                                NUMBER          11
    4 NULL
```

g. **Expense Table Structure**

```
UFI> SELECT * FROM COL WHERE TNAME = 'EXPENSE'

TNAME                                   COLNO  CNAME
                                COLTYPWIDTH WIDTH
  SCALE NULLS

EXPENSE                             1                   COST_FUN_NO
                                CHAR    4
      NOT NULL

EXPENSE                             2                   COST_CL_NO
                                CHAR    2
      NOT NULL

EXPENSE                                 3                       DT
                                DATE    8
      NOT NULL

EXPENSE                         4               ST              HOURS
                                NUMBER          7
    1 NULL

EXPENSE                         5               OT              HOURS
                                NUMBER          7
    1 NULL

EXPENSE                         6               ST              LABOR
                                NUMBER          11
    4 NULL
```

```
EXPENSE                        7          OT              LABOR
                               NUMBER     11
    4 NULL

EXPENSE                        8                          MATERIAL
                               NUMBER     11
    4 NULL

EXPENSE                        9                          OTHER
                               NUMBER     11
    4 NULL
```

h. **System Files**

1. System File Listing Structure

```
UFI> SELECT * FROM COL WHERE TNAME = 'FILES'

TNAME                          COLNO  CNAME
                               COLTYP WIDTH
  SCALE NULLS

FILES                          1                          FILEID
                               CHAR   15
      NULL

FILES                          2                          STRUCTYPE
                               CHAR   8
      NULL

FILES                          3                          FILETYPE
                               CHAR   8
      NULL
```

```
FILES                    4                    LOCATION
                       CHAR     8
        NULL

FILES                    5                    LASTMOD
                       CHAR     9
        NULL

FILES                    6                    COMMENTS
                       CHAR     70
        NULL
```

2. System File Listing

```
UFI> SELECT * FROM FILES;

FILEID      STRUCTYP FILETYPE LOCATION LASTMOD
 COMMENTS

BUDGET      TABLE    DATA     D DISK    11-DEC-86
 CONTAINS BUDGET FOR ST HOURS, OT HOURS, ST LABOR, OT LABOR,
MATERIAL AND OTHER

EXPENSE     TABLE    DATA     D DISK    11-DEC-86
 CONTAINS EXPENSES FOR HOURS, LABOR, MATERIAL AND OTHER BY
DATE

BCF         INDEX    DATA     D DISK    11-DEC-86
 INDEX BY COST FUNCTION FOR BUDGET TABLE

BCC         INDEX    DATA     D DISK    11-DEC-86
 INDEX BY COST CLASS FOR BUDGET TABLE

ECF         INDEX    DATA     D DISK    11-DEC-86
 INDEX BY COST FUNCTION FOR EXPENSE TABLE

ECC         INDEX    DATA     D DISK    11-DEC-86
```

i. **Programs and Modules**

1. Programs Listing Structure

```
UFI> SELECT * FROM COL WHERE TNAME = 'PROG';

TNAME                          COLNO  CNAME
                              COLTYPWIDTH
  SCALE NULLS

PROG                               1                PROGID
                                CHAR      14
       NULL

PROG                               2                FULLID
                                CHAR      35
       NULL

PROG                               3                LANGUAGE
                                CHAR      6
       NULL

PROG                               4                LASTMOD
                                CHAR      9
       NULL

PROG                               5                COMMENTS
                                CHAR      70
       NULL
```

2. Programs and Modules of the System

```
UFI> SELECT * FROM PROG;
```

PROGID FULLID LANGUA LASTMOD
COMMENTS

CCA COST CENTER ANALYSIS
C 11-DEC-86
MAIN MODULE CONTAINS THE THREE MODULES OF THE SYSTEM

GRAPHICS GRAPHIC DISPLAY OF DATA
C 11-DEC-86
ALLOWS USER TO USE DEVELOPED GRAPHS

COMDLEV COMMAND LEVEL
C 11-DEC-86
ALLOWS USER TO USE ORACLE AT THE COMMAND LEVEL

CCI COST CENTER INFORMATION
C 11-DEC-86
MAIN MENU DRIVEN SHELL FOR ORACLE

UFI USER FRIENDLY INTERFACE
C 11-DEC-86
ORACLE UTILITY

BUD_EXP BUDGET VS EXPENSES
C 11-DEC-86
DISPLAY AND COMPARISON ON BUDGET AND EXPENSE INFORMATION

INDVDISP INDIVIDUAL DISPLAY
C 11-DEC-86
BUDGET VS EXPENSES BY LABOR, MATERIAL OR OTHER

GETBUD BUDGET SUMMARY
C 11-DEC-86
DISPLAYS BUDGET BY COST FUNCTION/COST CLASS

TOBUDEXP TOTAL BUDGET VS EXPENSE
C 11-DEC-86
SUMS LABOR, MATERIAL AND OTHER FOR BUDGET AND EXPENSES

GETLAB DISPLAY LABOR
C 11-DEC-86
BUDGET VS EXPENSE BY COST FUNCTION/COST CLASS FOR LABOR

COMMENTS

GETHOUR DISPLAY HOURS
C 11-DEC-86
BUDGET VS EXPENSE BY COST FUNCTION/COST CLASS FOR HOURS

GETMAT DISPLAY MATERIAL
C 11-DEC-86
BUDGET VS EXPENSE BY COST FUNCTION/COST CLASS FOR MATERIAL

GETOTH DISPLAY OTHER
C 11-DEC-86
BUDGET VS EXPENSE BY COST FUNCTION/COST CLASS FOR OTHER

SELFUN SELECT FROM EMPLOYEE
ORACLE 11-DEC-86
SELECT FROM BUDGET BY COST FUNCTION

GETTOTF TOTAL BY COST FUNCTION
C 11-DEC-86
SUMS BUDGET AND EXPENSES BY COST FUNCTION

GETTOTC TOTAL BY COST CLASS
C 11-DEC-86
SUMS BUDGET AND EXPENSES BY COST CLASS

GETTOTFC TOTAL BY COST FUNCTION/COST CLASS
C 11-DEC-86
SUMS BUDGET AND EXPENSES BY COST FUNCTION/COST CLASS

GETSUM TOTAL BY COST CENTER
C 11-DEC-86
SUMS BUDGET AND EXPENSES FOR THE ENTIRE COST CENTER

SELHOUR SELECT FROM BUDGET
ORACLE 11-DEC-86
SELECT HOURS FROM BUDGET TABLE

SELLAB SELECT FROM BUDGET
ORACLE 11-DEC-86
SELECT LABOR FROM BUDGET TABLE

SELMAT SELECT FROM MATERIAL
ORACLE 11-DEC-86
SELECT MATERIAL FROM BUDGET TABLE

SELOTH SELECT FROM BUDGET
ORACLE 11-DEC-86
SELECT OTHER FROM BUDGET TABLE

SELEHOUR SELECT FROM EXPENSE
ORACLE 11-DEC-86
SELECT HOURS FROM EXPENSE TABLE SELELAB SELECT FROM EXPENSE ORACLE 11-DEC-86
SELECT LABOR FROM EXPENSE TABLE

SELEMAT SELECT FROM EXPENSE
ORACLE 11-DEC-86
SELECT MATERIAL FROM EXPENSE TABLE

SELEOTH SELECT FROM EXPENSE
ORACLE 11-DEC-86
SELECT OTHER FROM EXPENSE TABLE

SELBFUN SELECT FROM BUDGET
ORACLE 11-DEC-86
SELECT FROM BUDGET BY COST FUNCTION NO.

SELEFUN SELECT FROM EXPENSE
ORACLE 11-DEC-86
SELECT FROM EXPENSE BY COST FUNCTION NO.

SELBCL SELECT FROM BUDGET
ORACLE 11-DEC-86
SELECT FROM BUDGET BY COST CLASS

SELECL SELECT FROM EXPENSE
ORACLE 11-DEC-86
SELECT FROM EXPENSE BY COST CLASS

SELBCFCL SELECT FROM BUDGET
ORACLE 11-DEC-86
SELECT FROM BUDGET BY COST FUNCTION/COST CLASS

SELECFCL SELECT FROM EXPENSE
ORACLE 11-DEC-86
SELECT FROM EXPENSE BY COST FUNCTION/COST CLASS

SELSUM SELECT FROM BUDGET
ORACLE 11-DEC-86
SELECT TOTAL FROM BUDGET

SELSUMA SELECT FROM EXPENSE
ORACLE 11-DEC-86
SELECT TOTAL FROM EXPENSE

j. **Data Elements**

1. Data Elements Table Structure

UFI> SELECT * FROM COL WHERE TNAME = 'ELEMENTS';

TNAME COLNO CNAME
COLTYP WIDTH
SCALE NULLS

ELEMENTS 1 ELEMENTID
CHAR 20
NOT NULL

ELEMENTS 2 FULLID
CHAR 40
NULL

ELEMENTS 3 TYPE
CHAR 10
NULL

ELEMENTS 4 SOURCE
CHAR 15
NULL

ELEMENTS 5 UPDATEFREQ
CHAR 10
NULL

ELEMENTS 6 COMMENTS
CHAR 60
NULL

2. Data Elements Listing

UFI> SELECT * FROM ELEMENTS;

ELEMENTID FULLID TYPE SOURCE
UPDATEFREQ COMMENTS

COST_CL_NO COST CLASS NUMBER CHAR BUDGET
SELDOM SPECIFIES COST CLASS BY NUMBER

COST_CL_NO COST CLASS NUMBER CHAR EXPENSE
SELDOM SPECIFIES COST CLASS BY NUMBER

COST_FUNC_NO COST FUNCTION NUMBER CHAR BUDGET SELDOM SPECIFIES COST FUNCTION BY NUMBER

COST_FUNC_NO COST FUNCTION NUMBER CHAR EXPENSE SELDOM SPECIFIES COST FUNCTION BY NUMBER

DT EXPENSE DATA DATE DATE EXPENSE SEMIWEEKLY SPECIFIES THE DATE OF THE EXPENSE DATA

OTHOURS BUDGETED HOURS OVERTIME NUMBER BUDGET SEMIWEEKLY SPECIFIES HOURS BUDGETED FOR THE COST CENTER

STHOURS BUDGETED HOURS STRAIGHT TIME NUMBER BUDGET SEMIWEEKLY SPECIFIES HOURS BUDGETED FOR THE COST CENTER

OTHOURS EXPENSED HOURS OVERTIME NUMBER EXPENSE SEMIWEEKLY SPECIFIES HOURS EXPENSED BY THE COST CENTER

STHOURS EXPENSED HOURS STRAIGHT TIME NUMBER EXPENSE SEMIWEEKLY SPECIFIES HOURS EXPENSED BY THE COST CENTER

OTLABOR BUDGETED LABOR COSTS OVERTIME NUMBER BUDGET SEMIWEEKLY SPECIFIES LABOR BUDGETED FOR THE COST CENTER

STLABOR BUDGETED LABOR COSTS STRAIGHT TIME NUMBER BUDGET SEMIWEEKLY SPECIFIES LABOR BUDGETED FOR THE COST CENTER

OTLABOR EXPENSED LABOR COSTS OVERTIME NUMBER EXPENSE SEMIWEEKLY SPECIFIES LABOR EXPENSED FOR THE COST CENTER

STLABOR EXPENSED LABOR COSTS STRAIGHT TIME NUMBER EXPENSE SEMIWEEKLY SPECIFIES LABOR EXPENSED FOR THE COST CENTER

MATERIAL BUDGETED MATERIAL COSTS NUMBER BUDGET SEMIWEEKLY SPECIFIES MATERIAL BUDGETED FOR THE COST CENTER

MATERIAL EXPENSED MATERIAL COSTS NUMBER EXPENSE SEMIWEEKLY SPECIFIES MATERIAL EXPENSED FOR THE COST CENTER

OTHER BUDGETED OTHER COSTS NUMBER BUDGET SEMIWEEKLY SPECIFIES OTHER BUDGETED FOR THE COST CENTER

OTHER EXPENSED OTHER COSTS NUMBER EXPENSE SEMIWEEKLY SPECIFIES OTHER EXPENSED FOR THE COST CENTER

k. **System Element Hierarchy**

1. System Elements Hierarchy Table Structure

```
UFI> SELECT * FROM COL WHERE TNAME = 'CONT';
```

TNAME	COLNO	CNAME	COLTYPE	WIDTH	SCALE	NULLS
CONT	1	ID1	CHAR	15		NULL
CONT	2	TYPE1	CHAR	10		NULL
CONT	3	ID2	CHAR	16		NULL
CONT	4	TYPE2	CHAR	10		NULL

2. Systems Elements Hierarchy Listing

```
UFI> SELECT * FROM CONT;
```

ID1	TYPE1	ID2	TYPE2

EXPENSE	TABLE	COST_FUN_NO	ELEMENT
EXPENSE	TABLE	COST_CL_NO	ELEMENT
EXPENSE	TABLE	STHOURS	ELEMENT
EXPENSE	TABLE	STLABOR	ELEMENT
EXPENSE	TABLE	OTHOURS	ELEMENT
EXPENSE	TABLE	OTLABOR	ELEMENT
EXPENSE	TABLE	MATERIAL	ELEMENT
EXPENSE	TABLE	OTHER	ELEMENT
EXPENSE	TABLE	DT	ELEMENT
BUDGET	TABLE	COST_FUN_NO	ELEMENT
BUDGET	TABLE	COST_CL_NO	ELEMENT
BUDGET	TABLE	STHOURS	ELEMENT
BUDGET	TABLE	STLABOR	ELEMENT
BUDGET	TABLE	OTHOURS	ELEMENT
BUDGET	TABLE	OTLABOR	ELEMENT
BUDGET	TABLE	MATERIAL	ELEMENT
BUDGET	TABLE	OTHER	ELEMENT

3. System Process Table Structure

```
UFI> SELECT * FROM COL WHERE TNAME = 'PROCESS';
```

TNAME	COLNO	CNAME	COLTYP	WIDTH	SCALE	NULL
PROCESS	1	ID1	CHAR			
	20	NULL				
PROCESS	2	TYPE1	CHAR			
	10	NULL				
PROCESS	3	ID2	CHAR			
	20	NULL				

PROCESS	4	TYPE2	CHAR
	10	NULL	

4. Process Listing

```
UFI> SELECT * FROM PROCESS;
```

ID1	TYPE1	ID2	TYPE2
CCA	PROGRAM	GRAPHICS	PROGRAM
CCA	PROGRAM	COMDLEV	PROGRAM
CCA	PROGRAM	CCI	PROGRAM
GRAPHICS	PROGRAM	BAR	PROGRAM
GRAPHICS	PROGRAM	TRIPBAR	PROGRAM
GRAPHICS	PROGRAM	PLOT	PROGRAM
GRAPHICS	PROGRAM	COMBO	PROGRAM
COMDLEV	PROGRAM	UFI	PROGRAM
CCI	PROGRAM	BUD_EXP	PROGRAM
CCI	PROGRAM	EMPINFO	PROGRAM
CCI	PROGRAM	JOINFO	PROGRAM
BAR	PROGRAM	GRAF	FILE
TRIPBAR	PROGRAM	GRAF	FILE
PLOT	PROGRAM	BUD	FILE
PLOT	PROGRAM	GRAF1	FILE
COMBO	PROGRAM	GRAF	FILE
COMBO	PROGRAM	GRAF1	FILE
COMBO	PROGRAM	BUD	FILE
UFI	PROGRAM	EMPLOYEE	TABLE
UFI	PROGRAM	JO_EMP	TABLE
UFI	PROGRAM	JOB_ORD	TABLE
UFI	PROGRAM	COST_FUNC	TABLE
UFI	PROGRAM	COST_CLASS	TABLE
UFI	PROGRAM	BUDGET	TABLE
UFI	PROGRAM	EXPENSE	TABLE

BUD_EXP	PROGRAM	GETBUD	PROGRAM
BUD_EXP	PROGRAM	INDVDISP	PROGRAM
BUD_EXP	PROGRAM	TOTBUDEXP	PROGRAM
INDVDISP	PROGRAM	GETLAB	PROGRAM
INDVDISP	PROGRAM	GETMAT	PROGRAM
INDVDISP	PROGRAM	GETOTH	PROGRAM
INDVDISP	PROGRAM	GETHOUR	PROGRAM
GETBUD	PROGRAM	SELFUN	PROGRAM
TOTBUDEXP	PROGRAM	GETTOTF	PROGRAM
TOTBUDEXP	PROGRAM	GETTOTC	PROGRAM
TOTBUDEXP	PROGRAM	GETTOTCF	PROGRAM
TOTBUDEXP	PROGRAM	GETSUM	PROGRAM
GETLAB	PROGRAM	SELLAB	PROGRAM
GETLAB	PROGRAM	SELELAB	PROGRAM
GETHOUR	PROGRAM	SELHOUR	PROGRAM
GETHOUR	PROGRAM	SELEHOUR	PROGRAM
GETMAT	PROGRAM	SELMAT	PROGRAM
GETMAT	PROGRAM	SELEMAT	PROGRAM
GETOTH	PROGRAM	SELOTH	PROGRAM
GETOTH	PROGRAM	SELEOTH	PROGRAM
SELFUN	PROGRAM	BUDGET	TABLE
GETTOTF	PROGRAM	SELBFUN	PROGRAM
GETTOTF	PROGRAM	SELEFUN	PROGRAM
GETTOTC	PROGRAM	SELBCL	PROGRAM
GETTOTC	PROGRAM	SELECL	PROGRAM
GETTOTFC	PROGRAM	SELBCFCL	PROGRAM
GETTOTFC	PROGRAM	SELECFCL	PROGRAM
GETSUM	PROGRAM	SELSUM	PROGRAM
GETSUM	PROGRAM	SELSUMA	PROGRAM
SELHOUR	PROGRAM	BUDGET	TABLE
SELLAB	PROGRAM	BUDGET	TABLE
SELMAT	PROGRAM	BUDGET	TABLE
SELOTH	PROGRAM	BUDGET	TABLE
SELEHOUR	PROGRAM	EXPENSE	TABLE
SELELAB	PROGRAM	EXPENSE	TABLE

SELEMAT	PROGRAM	EXPENSE	TABLE
SELEOTH	PROGRAM	EXPENSE	TABLE
SELBFUN	PROGRAM	BUDGET	TABLE
SELEFUN	PROGRAM	EXPENSE	TABLE
SELBCL	PROGRAM	BUDGET	TABLE
SELECL	PROGRAM	EXPENSE	TABLE
SELBCFCL	PROGRAM	BUDGET	TABLE
SELECFCL	PROGRAM	EXPENSE	TABLE
SELSUM	PROGRAM	BUDGET	TABLE
SELSUMA	PROGRAM	EXPENSE	TABLE

5. UFI FILES

UFI files can be used to create tables in Oracle. These files allow the user to input the information using an editor rather than interactively with Oracle. The format that Oracle accepts new data is also shown with the INSERT command. Two samples of UFI files are shown below. Neither is complete as it stands. Only a small number of records that need to be read into the tables are shown. In actuality, every authorized cost function/cost class must have an entry for both budget and for each date of expense even if the values are all zeroes.

a. **BUDGET.UFI**

```
SYSTEM/MANAGER
SET ECHO OFF
SET VERIFY OFF
SET TERMOUT ON
SET SCAN OFF
CREATE TABLE BUDGET(COST_FUN_NO CHAR(4) NOT NULL,
   COST_CL_NO CHAR(2) NOT NULL,
   OTHOURS NUMBER(7,1),
   STHOURS NUMBER(7,1),
   OTLABOR NUMBER(11,4),
   STLABOR NUMBER(11,4),
   MATERIAL NUMBER(11,4),
   OTHER NUMBER(11,4));
SET SCAN ON
```

```
INSERT INTO BUDGET VALUES('9112', '02', '0',   '0',   '0',
    '0',        '0',        '0');
INSERT INTO BUDGET VALUES('9112', '03', '0',   '0',   '0',
    '0',        '0',        '0');
INSERT INTO BUDGET VALUES('9112', '04', '0',   '201',  '0',
    '3152',     '0',        '0');
INSERT INTO BUDGET VALUES('9112', '11', '0',   '0',   '0',
    '0',        '0',        '0');
INSERT INTO BUDGET VALUES('9112', '12', '0',   '0',   '0',
    '0',        '2000',     '4500');
INSERT INTO BUDGET VALUES('9112', '91', '414', '3000', '1567',
  '100000',     '0',        '0'); INSERT INTO BUDGET VALUES('9112', '93', '37',
'1800' '2000',
  '17540',      '0',        '0');
INSERT INTO BUDGET VALUES('9119', '43', '0',   '0',   '0',
    '0',        '0',        '0');
INSERT INTO BUDGET VALUES('9119', '54', '0',   '0',   '0',
    '0',        '507340',   '0');
INSERT INTO BUDGET VALUES('9119', '68', '0',   '0',   '0',
    '0',        '0',        '0');
COMMIT;
CREATE INDEX BCF ON BUDGET(COST_FUN_NO);
CREATE INDEX BCC ON BUDGET(COST_CL_NO);
UPDATE BUDGET SET LABOR = LABOR / 1000, MATERIAL = MATERIAL
/ 1000,
                    OTHER = OTHER / 1000;
GRANT SELECT ON BUDGET TO PUBLIC;
EXIT
```

b. **EXPENSE.UFI**

```
SYSTEM/MANAGER
SET ECHO OFF
SET VERIFY OFF
SET TERMOUT ON
SET SCAN OFF
CREATE TABLE EXPENSE(COST_FUN_NO CHAR(4) NOT NULL,
```

```
   COST_CL_NO CHAR(2) NOT NULL,
   DT DATE NOT NULL,
   OTHOURS NUMBER(7,1),
   STHOURS NUMBER(7,1),
   OTLABOR NUMBER(11,4),
   STLABOR NUMBER(11,4),
   MATERIAL NUMBER(11,4),
   OTHER NUMBER(11,4));
SET SCAN ON
INSERT INTO EXPENSE VALUES('9112', '02', '17-OCT-86', '     0',
'     0' , '        0', '        0'        '0',    '0');
INSERT INTO EXPENSE VALUES('9112', '03', '17-OCT-86', '     0',
         '0' , '        0', '       170',        '0',    '0');
INSERT INTO EXPENSE VALUES('9112', '04', '17-OCT-86', '    10',
        '80' , '      344', '      2344',       '2006',   '16');
INSERT INTO EXPENSE VALUES('9112', '11', '17-OCT-86', '     0',
'     1' , '        0', '        7',          '0',    '0');
INSERT INTO EXPENSE VALUES('9112', '12', '17-OCT-86', '     0',
'     0' , '        0', '        0',       '5725',   '929');
INSERT INTO EXPENSE VALUES('9112', '28', '17-OCT-86', '     0',
'     0' , '        0', '        0',          '0',   '0');
INSERT INTO EXPENSE VALUES('9112', '30', '17-OCT-86', '     0',


'     0' , '        0', '        0',          '0',   '4522');
INSERT INTO EXPENSE VALUES('9112', '33', '17-OCT-86', '     0',
'     0' , '        0', '        0',          '0',   '0');
INSERT INTO EXPENSE VALUES('9112', '39', '17-OCT-86', '     0',
'     0' , '        0', '        0',          '0',   '0');
INSERT INTO EXPENSE VALUES('9112', '43', '17-OCT-86', '     0',
'     0' , '        0', '        0',          '0',
INSERT INTO EXPENSE VALUES('9112', '54', '17-OCT-86', '     0',
'     0' , '        0', '        0',          '0',   '0');
INSERT INTO EXPENSE VALUES('9112', '68', '17-OCT-86', '     0',
'     0' , '        0', '        0',          '0',   '0');
```

```
INSERT INTO EXPENSE VALUES('9112', '91', '17-OCT-86', '    12',
'   140 , '      500', '     4089',       '0',  '0');
INSERT INTO EXPENSE VALUES('9112', '93', '17-OCT-86', '     0',
'    56' , '       0', '      593',       '0',  '0');
INSERT INTO EXPENSE VALUES('9112', '96', '17-OCT-86', '.    0',
'     0' , '       0', '       0',       '0', '498752');
INSERT INTO EXPENSE VALUES('9112', '97', '17-OCT-86', '     0',
'     0' , '       0', '       0',       '0',  '0');
INSERT INTO EXPENSE VALUES('9112', '98', '17-OCT-86', '     0',
'     0' , '       0', '       0',      '5235',
INSERT INTO EXPENSE VALUES('9112', '99', '17-OCT-86', '     0',
'     0' , '       0', '       0',       '0',  '0');
INSERT INTO EXPENSE VALUES('9113', '02', '17-OCT-86', '     0',
'     0' , '       0', '       0',       '0',  '0');
INSERT INTO EXPENSE VALUES('9113', '03', '17-OCT-86', '     0',
'     0' , '       0', '       0',       '0',  '0');
INSERT INTO EXPENSE VALUES('9113', '04', '17-OCT-86', '     0',
'     0' , '       0', '       0',       '0',  '0');
INSERT INTO EXPENSE VALUES('9113', '11', '17-OCT-86', '     0',
'     0' , '       0', '       0',       '0',  '0');
INSERT INTO EXPENSE VALUES('9113', '12', '17-OCT-86', '     0',
'     0' , '       0', '       0',       '0',  '0');
INSERT INTO EXPENSE VALUES('9113', '23', '17-OCT-86', '     0',
'     0' , '       0', '       0',       '0',  '0');
INSERT INTO EXPENSE VALUES('9113', '28', '17-OCT-86', '     0',
'     0' , '       0', '       0',       '0',  '0');
INSERT INTO EXPENSE VALUES('9113', '30', '17-OCT-86', '     0',
'     0' , '       0', '       0',       '0',  '0');
INSERT INTO EXPENSE VALUES('9113', '32', '17-OCT-86', '     0',
'     0' , '       0', '       0',       '0',  '0');
INSERT INTO EXPENSE VALUES('9113', '39', '17-OCT-86', '     6',
'    10' , '      31', '      100',       '0',  '0');
INSERT INTO EXPENSE VALUES('9113', '43', '17-OCT-86', '     0',
'     0' , '       0', '       0',       '0',  '0');
INSERT INTO EXPENSE VALUES('9113', '54', '17-OCT-86', '     0',
```

```
'    0', '        0', '        0',        '0',  '0');
INSERT INTO EXPENSE VALUES('9113', '68', '17-OCT-86', '    0',
'    0', '        0', '        0',        '0',  '0');
INSERT INTO EXPENSE VALUES('9113', '91', '17-OCT-86', '   67',
'  500', '      400', '    14026',        '0',  '0');
INSERT INTO EXPENSE VALUES('9113', '92', '17-OCT-86', '  531',
' 2000', '     1599', '    43000',        '0', '-594');
INSERT INTO EXPENSE VALUES('9113', '93', '17-OCT-86', '   64',
'  800', '     2383', '    10000',        '0',  '0');
INSERT INTO EXPENSE VALUES('9113', '96', '17-OCT-86', '    0',
'    0', '        0', '        0',        '0',  '0');
INSERT INTO EXPENSE VALUES('9113', '99', '17-OCT-86', '    0',
'    0', '        0', '        0',        '0',  '0');
INSERT INTO EXPENSE VALUES('9114', '04', '17-OCT-86', '    0',
'    0', '        0', '        0',        '0',  '0');
INSERT INTO EXPENSE VALUES('9114', '12', '17-OCT-86', '    0',
'    0', '        0', '        0',        '0',  '0');
INSERT INTO EXPENSE VALUES('9114', '28', '17-OCT-86', '    0',
'    0', '        0', '        0',        '0',  '0');
INSERT INTO EXPENSE VALUES('9114', '30', '17-OCT-86', '    0',
'    0', '        0', '        0',        '0',  '0');
INSERT INTO EXPENSE VALUES('9114', '32', '17-OCT-86', '    0',
'    0', '        0', '        0',        '0',  '0');
INSERT INTO EXPENSE VALUES('9114', '33', '17-OCT-86', '    0',
'    0', '        0', '        0',        '0',  '0');
INSERT INTO EXPENSE VALUES('9114', '43', '17-OCT-86', '    0',
'    0', '        0', '        0',        '0',  '0');
INSERT INTO EXPENSE VALUES('9114', '54', '17-OCT-86', '    0',
'    0', '        0', '        0',        '0',  '0');
INSERT INTO EXPENSE VALUES('9114', '68', '17-OCT-86', '    0',
'    0', '        0', '        0',        '0',  '0');
INSERT INTO EXPENSE VALUES('9114', '94', '17-OCT-86', '    0',
'    0', '        0', '        0',        '0', '4968');
INSERT INTO EXPENSE VALUES('9114', '95', '17-OCT-86', '    0',
'    0', '        0', '        0',        '0', '246');
```

```
INSERT INTO EXPENSE VALUES('9114', '98', '17-OCT-86', '    0',
'    0', '    0', '    0',    '0',  '0');
INSERT INTO EXPENSE VALUES('9115', '02', '17-OCT-86', '    0',
'    0', '    0', '    0',    '0',  '0');
INSERT INTO EXPENSE VALUES('9115', '03', '17-OCT-86', '    0',
'    0', '    0', '    0',    '0',  '0');
INSERT INTO EXPENSE VALUES('9115', '04', '17-OCT-86', '    0',
'    0', '    0', '    0',    '0',  '0');
INSERT INTO EXPENSE VALUES('9115', '11', '17-OCT-86', '    0',
'    0', '    0', '    0',    '0',  '0');
INSERT INTO EXPENSE VALUES('9115', '12', '17-OCT-86', '    0',
'    0', '    0', '    0',    '0',  '0');
INSERT INTO EXPENSE VALUES('9115', '28', '17-OCT-86', '    0',
'    0', '    0', '    0',    '0',  '0');
INSERT INTO EXPENSE VALUES('9115', '30', '17-OCT-86', '    0',
'    0', '    0', '    0',    '0',  '0');
INSERT INTO EXPENSE VALUES('9115', '32', '17-OCT-86', '    0',
'    0', '    0', '    0',    '0',  '0');
INSERT INTO EXPENSE VALUES('9115', '33', '17-OCT-86', '    0',
'    0', '    0', '    0',    '0',  '0');
INSERT INTO EXPENSE VALUES('9115', '39', '17-OCT-86', '    0',
'    0', '    0', '    0',    '0',  '0');
INSERT INTO EXPENSE VALUES('9115', '43', '17-OCT-86', '    0',
'    0', '    0', '    0',    '0',  '0');
INSERT INTO EXPENSE VALUES('9115', '54', '17-OCT-86', '    0',
'    0', '    0', '    0',    '0',  '0');
INSERT INTO EXPENSE VALUES('9115', '68', '17-OCT-86', '    0',
'    0', '    0', '    0',    '0',  '0');
INSERT INTO EXPENSE VALUES('9115', '91', '17-OCT-86', '    0',
'  103', '    0', '  2205',    '0',  '0');
INSERT INTO EXPENSE VALUES('9115', '93', '17-OCT-86', '    3',
'   90', '   50', '   703',    '0',  '0');
INSERT INTO EXPENSE VALUES('9116', '02', '17-OCT-86', '    0',
'    0', '    0', '    0',    '0',  '0');
```

```
INSERT INTO EXPENSE VALUES('9116', '03', '17-OCT-86', '     0',
'     0' , '       0', '       0',        '0',  '0');
INSERT INTO EXPENSE VALUES('9116', '04', '17-OCT-86', '     4',
'    60' , '      24', '     900',        '0',  '0');
INSERT INTO EXPENSE VALUES('9116', '11', '17-OCT-86', '     0',
'     0' , '       0', '       0',        '0',  '0');
INSERT INTO EXPENSE VALUES('9116', '12', '17-OCT-86', '     0',
'     0' , '       0', '       0',        '0',  '0');
INSERT INTO EXPENSE VALUES('9116', '28', '17-OCT-86', '     0',
'     0' , '       0', '       0',        '0',  '0');
INSERT INTO EXPENSE VALUES('9116', '30', '17-OCT-86', '     0',
'     0' , '       0', '       0',        '0',  '0');
INSERT INTO EXPENSE VALUES('9116', '32', '17-OCT-86', '     0',
'     0' , '       0', '       0',        '0',  '0');
INSERT INTO EXPENSE VALUES('9116', '33', '17-OCT-86', '     0',
'     0' , '       0', '       0',        '0',  '0');
INSERT INTO EXPENSE VALUES('9116', '39', '17-OCT-86', '     0',
'     0' , '       0', '       0',        '0',  '0');
INSERT INTO EXPENSE VALUES('9116', '43', '17-OCT-86', '     0',
'     0' , '       0', '       0',        '0',  '0');
INSERT INTO EXPENSE VALUES('9116', '54', '17-OCT-86', '     0',
'     0' , '       0', '       0',        '0',  '0');
INSERT INTO EXPENSE VALUES('9116', '68', '17-OCT-86', '     0',
'     0' , '       0', '       0',        '0',  '0');
INSERT INTO EXPENSE VALUES('9116', '91', '17-OCT-86', '     5',
'    70' , '      93', '    1400',        '0',  '0');
INSERT INTO EXPENSE VALUES('9116', '93', '17-OCT-86', '   100',
'   718' , '     473', '   11000',        '0',  '0');
INSERT INTO EXPENSE VALUES('9117', '02', '17-OCT-86', '     0',
'     0' , '       0', '       0',        '0',  '0');
INSERT INTO EXPENSE VALUES('9117', '03', '17-OCT-86', '     0',
'     0' , '       0', '       0',        '0',  '0');
INSERT INTO EXPENSE VALUES('9117', '04', '17-OCT-86', '     0',
'     0' , '       0', '       0',        '0',  '0');
INSERT INTO EXPENSE VALUES('9117', '11', '17-OCT-86', '     0',
```

' 0', ' 0', ' 0', '0', '0');
INSERT INTO EXPENSE VALUES('9117', '12', '17-OCT-86', ' 0',
' 0', ' 0', ' 0', '0', '0');
INSERT INTO EXPENSE VALUES('9117', '23', '17-OCT-86', ' 0',
' 0', ' 0', ' 0', '0', '0');
INSERT INTO EXPENSE VALUES('9117', '28', '17-OCT-86', ' 0',
' 0', ' 0', ' 0', '0', '0');
INSERT INTO EXPENSE VALUES('9117', '30', '17-OCT-86', ' 0',
' 0', ' 0', ' 0', '0', '0');
INSERT INTO EXPENSE VALUES('9117', '32', '17-OCT-86', ' 0',
' 0', ' 0', ' 0', '0', '0');
INSERT INTO EXPENSE VALUES('9117', '33', '17-OCT-86', ' 0',
' 0', ' 0', ' 0', '0', '0');
INSERT INTO EXPENSE VALUES('9117', '39', '17-OCT-86', ' 0',
' 0', ' 0', ' 0', '0', '0');
INSERT INTO EXPENSE VALUES('9117', '43', '17-OCT-86', ' 0',
' 0', ' 0', ' 0', '0', '0');
INSERT INTO EXPENSE VALUES('9117', '54', '17-OCT-86', ' 0',
' 0', ' 0', ' 0', '0', '0');
INSERT INTO EXPENSE VALUES('9117', '68', '17-OCT-86', ' 0',
' 0', ' 0', ' 0', '0', '0');
INSERT INTO EXPENSE VALUES('9117', '91', '17-OCT-86', ' 23',
' 200', ' 78', ' 3800', '0', '0');
INSERT INTO EXPENSE VALUES('9117', '93', '17-OCT-86', ' 61',
' 1200', ' 190', ' 16000', '0', '0');
INSERT INTO EXPENSE VALUES('9117', '95', '17-OCT-86', ' 0',
' 0', ' 0', ' 0', '0', '0');
INSERT INTO EXPENSE VALUES('9117', '96', '17-OCT-86', ' 0',
' 0', ' 0', ' 0', '0', '0');
INSERT INTO EXPENSE VALUES('9117', '97', '17-OCT-86', ' 0',
' 0', ' 0', ' 0', '9654', '0');
INSERT INTO EXPENSE VALUES('9117', '99', '17-OCT-86', ' 0',
' 0', ' 0', ' 0', '0', '0');
INSERT INTO EXPENSE VALUES('9118', '02', '17-OCT-86', ' 0',
' 0', ' 0', ' 0', '0', '0');

```
INSERT INTO EXPENSE VALUES('9112', '02', '31-OCT-86', '     0',
'     0', '        0', '        0',        '0',  '0');
INSERT INTO EXPENSE VALUES('9112', '03', '31-OCT-86', '     0',
'     0', '        0', '        0',        '0',  '0');
INSERT INTO EXPENSE VALUES('9112', '04', '31-OCT-86', '   181',
'     0', '     4711', '        0',      '2006',  '16');
INSERT INTO EXPENSE VALUES('9112', '11', '31-OCT-86', '     6',
'     0', '       75', '        0',        '0',  '0');
INSERT INTO EXPENSE VALUES('9112', '12', '31-OCT-86', '     0',
'     0', '        0', '        0',     '6280',   '929');
INSERT INTO EXPENSE VALUES('9112', '28', '31-OCT-86', '     0',
'     0', '        0', '        0',        '0',  '0');
INSERT INTO EXPENSE VALUES('9112', '30', '31-OCT-86', '     0',
'     0', '        0', '        0',        '0',  '5950');
INSERT INTO EXPENSE VALUES('9112', '33', '31-OCT-86', '     0',
'     0', '        0', '        0',        '0',  '0');
INSERT INTO EXPENSE VALUES('9112', '43', '31-OCT-86', '     0',
'     0', '        0', '        0',        '0',  '0');
INSERT INTO EXPENSE VALUES('9119', '43', '31-OCT-86', '     0',
'     0', '        0', '        0',        '0',  '0');
INSERT INTO EXPENSE VALUES('9119', '54', '31-OCT-86', '     0',
'     0', '        0', '        0',        '0',  '0');
INSERT INTO EXPENSE VALUES('9119', '68', '31-OCT-86', '     0',
'     0', '        0', '        0',        '0',  '0');
COMMIT;
CREATE INDEX ECF ON EXPENSE(COST_FUN_NO);
CREATE INDEX ECC ON EXPENSE(COST_CL_NO);
UPDATE EXPENSE SET LABOR = LABOR / 1000, MATERIAL = MATERIAL
/ 1000,
OTHER = OTHER / 1000;
GRANT SELECT ON EXPENSE TO PUBLIC;
EXIT
```

APPENDIX B
COST CENTER ANALYSIS USER MANUAL (MINICOMPUTER)

1. INTRODUCTION

This system is designed to allow Cost Center managers the ability to track expenses and compare them to budgeted amounts. In that way users can gain greater insight into costs and the reasons behind them. This ability should give managers a much clearer appreciation of where and how costs are being produced.

Graphic display of some of the numerical output is provided using TEL-A-GRAF business graphic system.

2. REQUIREMENTS

Cost Center Analysis (minicomputer) is designed for a Prime 9755 computer with on-line capabilities. The software is written in CPL, Prime's Command Processor Language, and the TEL-A-GRAF command language. The software requirements are the CCA (minicomputer) program, TEL-A-GRAF, and the PRIMOS operating system.

3. STARTING THE SYSTEM

At the PRIMOS command prompt (OK, or ER,) type < R CCA > (do not type the less than (<) or greater than (>) symbols) to activate the Cost Center Analysis program. The "R" stands for RESUME. RESUME allows the user to interactively run a CPL program. The operating system then looks for a "compiled and loaded program." If it does not find the file, it will then look for the appropriate CPL file. The CPL interpreter will then act on the program CCA.CPL and issue the appropriate instructions to PRIMOS [Ref. 14: p. 1- 1-2].

The program's top level menu should then display. If it does not, check to see that you are at the PRIMOS command prompt. If so, follow the above steps to load CCA. Ensure you type the "R" before CCA.

4. MAIN MENU

With this version you are allowed two options (see Figure B.1). In future versions more options may be added.

At the "Select One:" prompt you may respond with "T" or "Q" to identify your desired option (see figure B.1). The system will validate your response, so the select

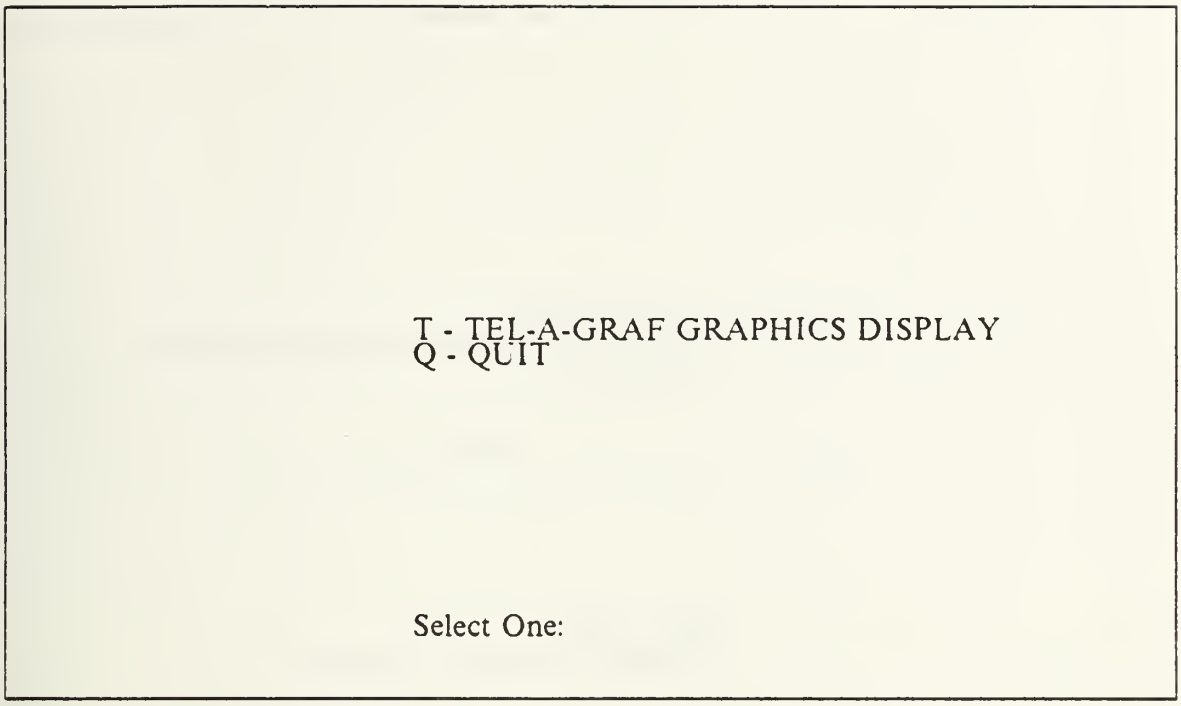


Figure B.1 CCA Top Level Menu.

prompt will reappear if anything other than "T" or "Q" are entered. If you enter "Q" you will leave CCA and be returned to the PRIMOS level at the command prompt (OK,). Option "T" will select the TEL-A-GRAF option, allowing you to produce various graphs from standard formatting files. CCA.

After the system validates your response (T), you will be asked if you wish to go directly to TEL-A-GRAF (see Figure B.2).

This gives you the option of entering TEL-A-GRAF immediately, without selecting one of the standard graphs that are provided the user through CCA. Entering TEL-A-GRAF at this level requires you to know the command language, and how to format your data and include files. The include files contain the programs which TEL-A-GRAF uses to produce the graphs. The data files contain the formatted data. The novice or occasional user is cautioned to use the graphic formats provided by CCA. The structure of the data and include files will be dealt with in more detail below.

If you do not wish to go directly to TEL-A-GRAF, select "N" or "NO". This query will only accept a yes or no response.

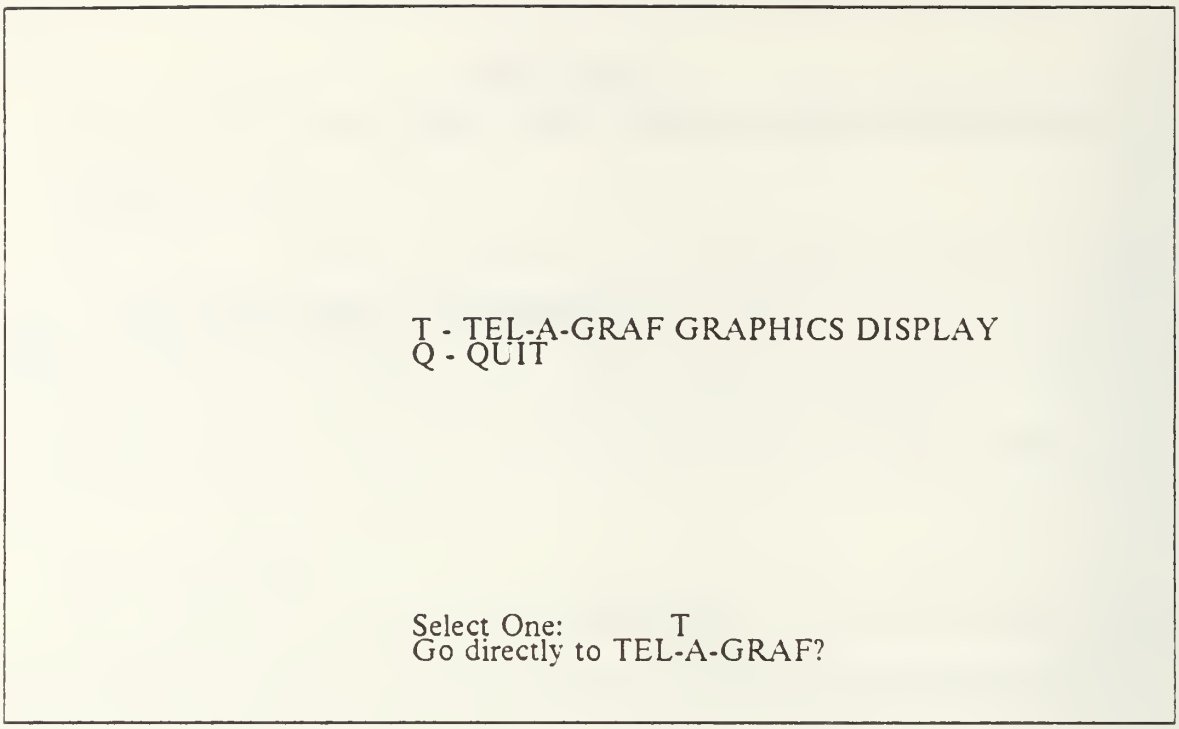


Figure B.2 Prompt for the user's response.

5. CHOOSE COST CENTER MENU

This menu allows the user to select the particular Cost Center he wishes to investigate (see figure B.3). In this version only one Cost Center is provided. Future versions could easily incorporate more.

At the "Select One:" prompt enter "1." This is the only response that CCA will validate. An inappropriate response will cause the prompt to reappear.

6. GRAPH PLOT CODE MENU

This menu allows the user to select the type of standard graph he desires (see figure B.4).

Option "A" gives the user the opportunity to produce a plot of budget to expenses. Within the plot a bar chart of the budget data is overlayed. This graph provides the capability to study expense to budget variances, and to quickly identify how closely to budget the Cost Center is tracking.

Option"B" produces a bar chart of the expense to budget data. It displays that information in "time elapsed." This allows the user to identify variances and also the budget amount that should be expensed at the particular data date.

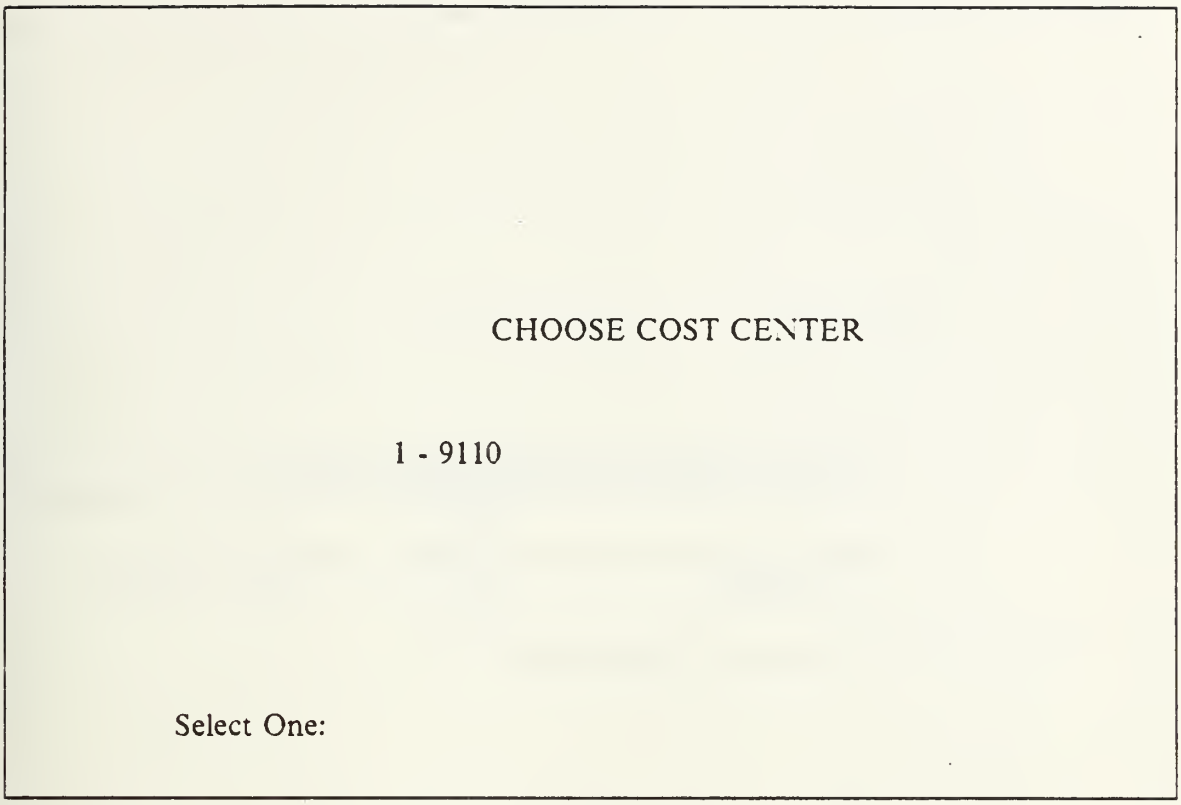


Figure B.3 Prompt for the user's response.

Option "C" develops a composite of four variance graphs: Percent Expended, Data Normalized on Percent Elapsed Time, Variance in Dollars, and Percent Variance. These bar charts show whether the particular expenses accrued have positive or negative variances and their magnitudes.

7. PLOT OPTIONS MENU

This menu allows the user to further define the particular graph selected (see figure B.5).

Option "A" causes the graph produced to display the total budget and expenses of the Cost Center selected. Bar Graphs will be broken down by Cost Functions within the Cost Center.

Options "2" through "9" will provide the data by particular Cost Function. On the bar graphs the Cost Functions will be broken down by Cost Classes.

All expense data is of the most current date entered into the updated data files.

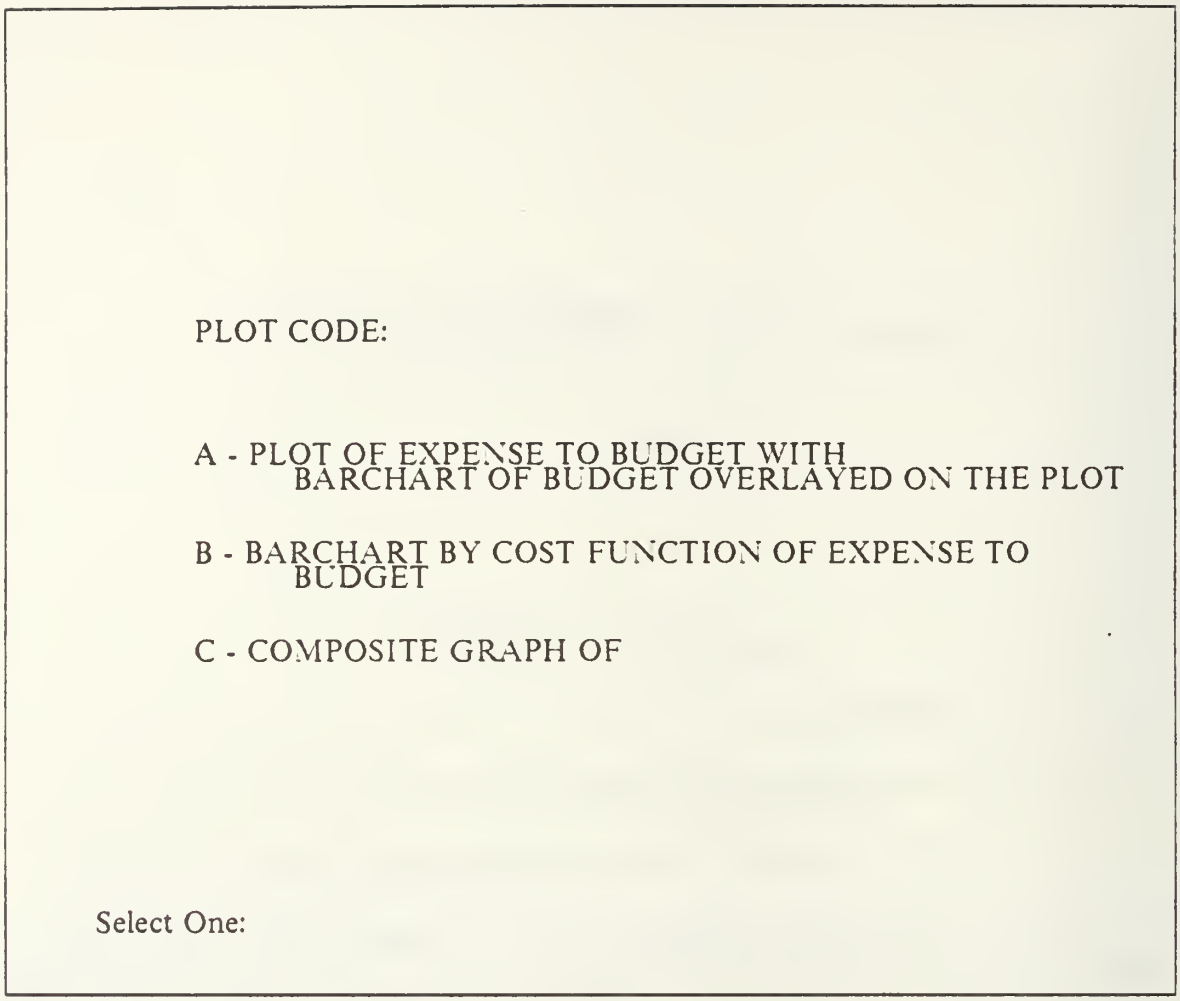


Figure B.4 Plot Code Selections.

8. ENTER TEL-A-GRAF

At this point CCA will open TEL-A-GRAF and issue the appropriate commands to produce the desired graph. The commands will scroll by on the screen and then you will get a blank screen. In a few seconds the selected graph will be drawn on the screen. After it is complete you may study the graph. To continue strike the <ENTER> or <RETURN> key. You will be returned to the command language level of TEL-A-GRAF. You may continue working at this level, if you know the appropriate commands. To return to CCA, type <QUIT.>. You will then receive the prompt in figure B.6 Typing "N" or "NO" will allow you to produce another graph through CCA. Typing "Y" or "YES" will return you to the PRIMOS command prompt.

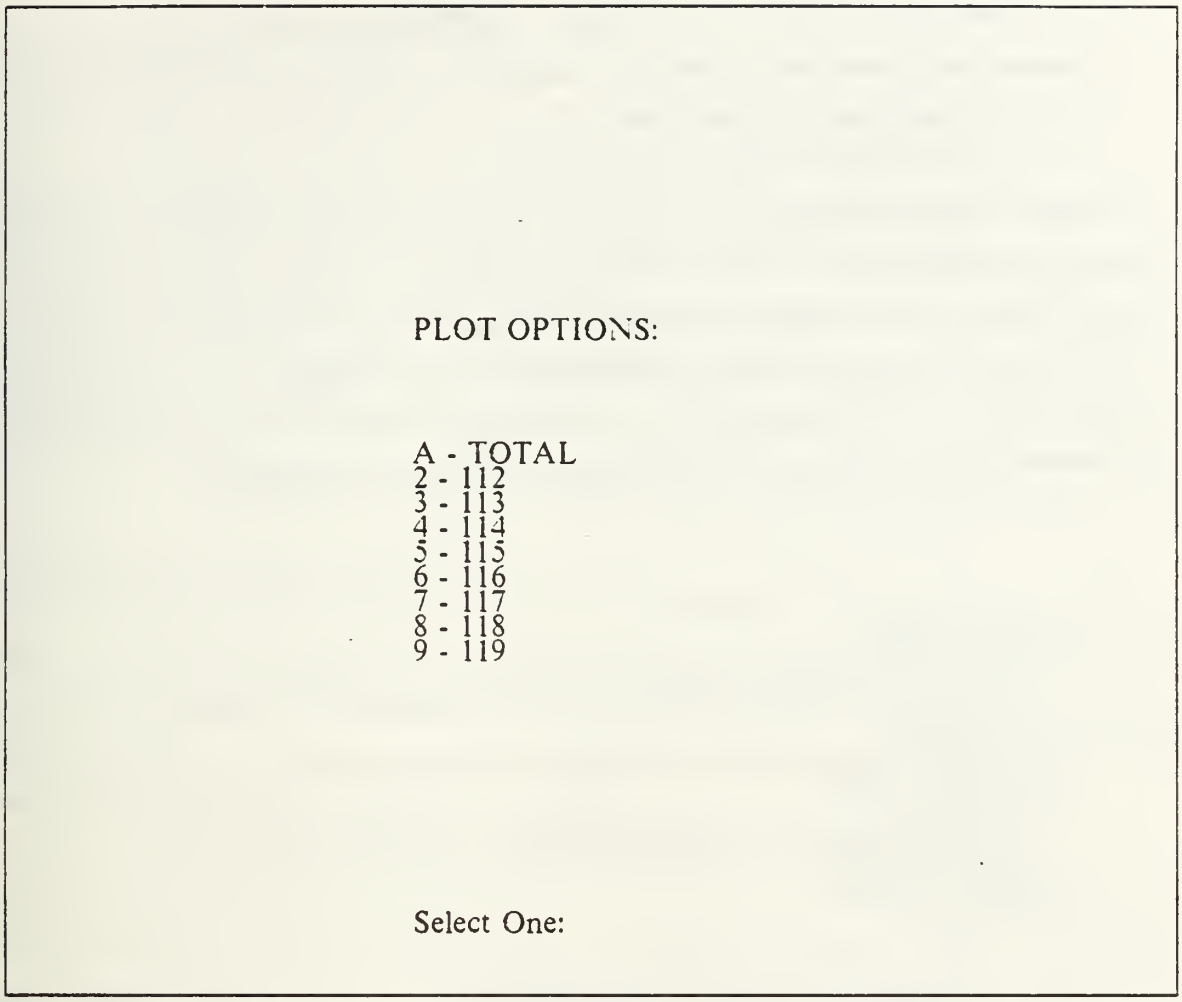


Figure B.5 Plot Option Selections.

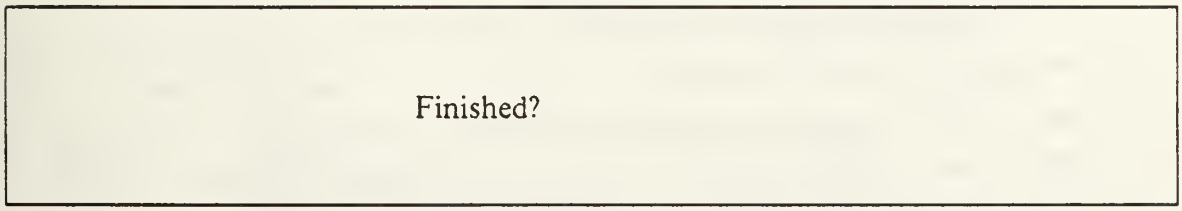


Figure B.6 Completion Query.

9. USING TEL-A-GRAF

TEL-A-GRAF is a very powerful graphics system. With this power comes many options and different ways of accomplishing the same tasks. We shall discuss only a

few of them here. The TEL-A-GRAF User Manual [Ref. 12], will answer most questions you may have if you want to become more familiar with TEL-A-GRAF.

a. **Making Your Own Data Files**

Many methods exist for entering data for a graph. We shall talk of only two methods. The first is creating the data file from the editor, and the second is inputing the data while in TEL-A-GRAF. All data must be inputed in millions of dollars for a Cost Center. For example, five hundred thousand dollars, $500,000.00 is represented as .5 and one million dollars, $1,000,000.00, is represented as 1.0. When data is inputed for a Cost Function, by Cost Class, it must be in thousands of dollars. Variance data is in Dollars or decimal representation for percentages.

```
INPUT DATA.
"BUDGET"
  1 0.77287 2 2.13234 3 4.33018 4 0.053063 5 0.274093 6 1.48263 7  --
0.0898 8 0.507
"BUDGET%"
  1 0.60284 2 1.66322 3 3.37754 4 0.04139 5 0.21379 6 1.15645 7 --
0.07005 8 0.39546
"EXPENSES"
  1 0.69411 2 1.3034 3 3.31009 4 0.04487 5 0.21733 6 1.10264 7 --
0.0665 8 0.17234
END OF DATA.
**FILE**
```

Figure B.7 TEL-A-GRAF Data File for Triple Bar Chart.

1. Creating a Data File From the Editor.

Creating data files from the editor offers several advantages to entering the data interactively with TEL-A-GRAF. First, by entering the data in the editor mode, you can check the data for errors that may have occurred when entering the data. Secondly, if several different sets of data are to be graphed prepositioning the data in files will shorten the amount of time you will need to be at the graphics terminal. Lastly, if the data will be used later, you will not have to input it again.

This is not a tutorial on how to use the editor. For information on the editors available and how to use them see the Prime Computer Training Manual [Ref. 15]

Figure B.7 is an example of a typical data file. This particular data was used with the Triple Bar chart. However, the format is the same for all graphs. The first line must be INPUT DATA. The period at the end of the line is very important so do not forget it. The next line should be the name of the data. This name will appear in any legend that you may want to produce. Next comes the actual data. The data must be in X, Y pairs, the independent followed by the dependent. The comma between the x,y is optional, a space will do. The data may also be written in column form which makes changes and error checking much easier. In the include files we shall discuss, all dependent values are 0 at the origin, followed by 1, 2, 3, 4, etc. The label for each however, is not 1, 2, ,3 etc. The labels have been given other names such as the Cost Functions 9112, 9113, etc. The corresponding position is the number that must be in the data file, not the label.

If more than one set of data is to be graphed on that graph, the remaining data can be entered in the same way. When all data is entered, the last statement must be END OF DATA. The last statement shown in figure B.7 is the end of file symbol for TEL-A-GRAF. This is optional for the user to put in because TEL-A-GRAF will automatically write it in the file after it is done.

Without the data base implemented you may think that this system is useless. You can, however, enter your own data into the appropriate file to use one of the graphs shown. Figure B.8 shows which data goes with which graphs. By entering the updated information in the appropriate data file in the appropriate format, the CCA menu system can be used with no knowledge of TEL-A-GRAF. Only knowing how to build a data file is required.

Data File	*Graph*
B110	Budget Bar Graph in corner of composite graph
BBE110	Triple Bar Graph
BE110	Plot of budget vs expenses
PB110	Percent Expended Bar Graph for variance analysis
NB110	Data Normalized on Precent Elapsed Time
VB110	Variance in Dollars
PV110	Percent Variance

Figure B.8 Name of Data Files Matched With the Appropriate Graphs.

The name of the data file consists of two parts, letters followed by numbers. The letters may represent on of two things. In the composite graph, the bar graph of the budget is linked with the data file B110. No matter which Cost Function you select for the plot in the menu, the budget for the Cost Center broken down by Cost Function is displayed in the corner. All other graphs are linked as shown is figure B.8 . When graphing a Cost Function instead of the cost center, the data file's numbers are the Cost Function number, with the exception of the b110 series, which always references b110. For example, the data file linked to the triple bar graph for Cost Function 9112 is BBE112, for Cost Function 9113 is BBE113, and so on. This convention holds for all classes of data file names.

2. Entering Data From Within TEL-A-GRAF

This process is more complicated than using the editor, and has a greater potential for errors getting by. A knowledge of TEL-A-GRAF is necessary in order to use this method. Either your own commands must be issued or, the files already created can be included. Data is entered the same way as it is in figure B.7 , and as explained in the previous section.

Since a knowledge of TEL-A-GRAF is necessary, this procedure assumes a more sophisticated user. Modifying the existing graphs to suit your own needs and using different data may be a useful technique for learning TEL-A-GRAF.

b. **TEL-A-GRAF Commands**

This section is designed to show you how to manipulate existing files that TEL-A-GRAF uses. It is not designed to show you how to write original TEL-A-GRAF programs. Figure B.9 list the names of the programs and their relation with other modules.

The main module for each graph contains the commands to generate a graph from Cost Center 9110. The related graphs modify this basic graph to get the appropriate labels and titles.

1. Using Existing Files

Existing files are brought in to TEL-A-GRAF through the use of the INCLUDE command. The include command brings in a file which is then processed. If more than one file is brought in, the first is processed, and changes or additions to that file are made when the next include file is processed.

An example of a possible command level interaction follows. Suppose you have created a data file named 'BBE112' for a triple bar graph of Cost Function 9112

Graph	*Module*	*Related Modules*
Bar graph of Budget	B1	none
Plot of Budget vs Expense	EX2	EX112, EX113, EX114, EX115 EX116, EX117, EX118, EX119
Triple bar graph	B4	B112, B113, B114, B115, B116, B117, B118, B119
Percent Expended	PERBAR	none
Normalized on Time	NORBAR	none
Variance in Dollars	VARBAR	none
Percent Variance	PERVAR	none

Figure B.9 Graphic Program Module Relations.

broken down by Cost Class. The data file must contain a value for each authorized Cost Class, whether it has been used of not.

The first prompt from TEL-A-GRAF is SPECIFY FILES.< RETURN > is the appropriate response. Next you need to specify the data file. Next you must include the first include file. This is the main module for the graph. For this example it is 'B4'. Next you must specify the second include file to set the appropriate headings and labels. In this example that file is B112. Figure B.10 demonstrates the proper sequence for this example.

```
SPECIFY FILES:    < RETURN >
GENERATE LEVEL..ENTER:
< DATAFILE IS 'BBE112'. >
GENERATE LEVEL..ENTER:
< INCLUDE 'B4'. >
INCLUDE FILE BEING PROCESSED
ENTER MORE OR PERIOD:
< INCLUDE 'B112'.>
ENTER MORE OR PERIOD:
< GO. >
```

Figure B.10 Interactive Session with TEL-A-GRAF.

2. Appending Existing Files

If you went through the process of figure B.9 , but you were not satisfied, you can make changes from inside TEL-A-GRAF. First you must type "CONTINUE." This allows you to continue with the same graph. Now you can change the data, the title, or anything else you wish. You can change the data file either by specifying another data file as before, or by inputing the data by hand as described in the data section. Changes in the title can be made by issuing commands such as "TITLE IS 'BUDGET VS EXPENSES FOR COST FUNCTION 9112'." This will change the title to whatever you write.

For the user who wants to use TEL-A-GRAF at this level, further information is available in the user manual [Ref. 12].

APPENDIX C
COST CENTER ANALYSIS USER MANUAL (MICROCOMPUTER)

1. INTRODUCTION

This system is designed to allow Cost Center managers the ability to track expenses and compare them to budgeted amounts. Additionally, the user can identify the jobs that have accrued these expenses. In that way users can gain greater insight into those costs and the reasons behind them. This ability should give managers a much clearer appreciation of where and how costs are being produced.

In addition to the query screen responses, the user can receive hardcopy responses. Graphical display of some of the numerical output is also provided using the systems graphic utilities.

2. REQUIREMENTS

Cost Center Analysis hardware requirements are an IBM PC/XT/AT with at least 640K and a hard disk. A printer is optional for the output print options. The software requirements are the Oracle Data Base Management System (DBMS), PC/MS-DOS, and the Cost Center Analysis and Graphic Utilities programs, all installed on the hard disk. In order to support CCA and the graphics utilities the following utilities are required as well:

The CUL library from Essential Software Incorporated.[4]

GraphiC from Scientific Endeavours.[5]

3. STARTING THE SYSTEM

The first step to begin the CCA program is to turn the computer on. This is done by turning the switch on the power board. At the DOS command line type "Oracle" to activate the Oracle DBMS. This system is essential to the operation of Cost Center Analysis. Version 1.0 will not call Oracle first, due to the memory requirements. It is hoped in later versions that this service will be provided for the user.

---

[4] *C Utilities Library User Guide* (*Version 2.0*) ESI, Maplewood, NJ 07040, 1985.

[5] Rome, James A. and George G. Kelley, *GraphiC Version 2.1*, Scientific Endeavours, Rte. 4, Box 79, Kingston, TN 37763, 1985.

Once Oracle has displayed its logo and licensing information, type "CCA" to load the Cost Center Analysis system. This places the user at the main menu.

4. MAIN MENU

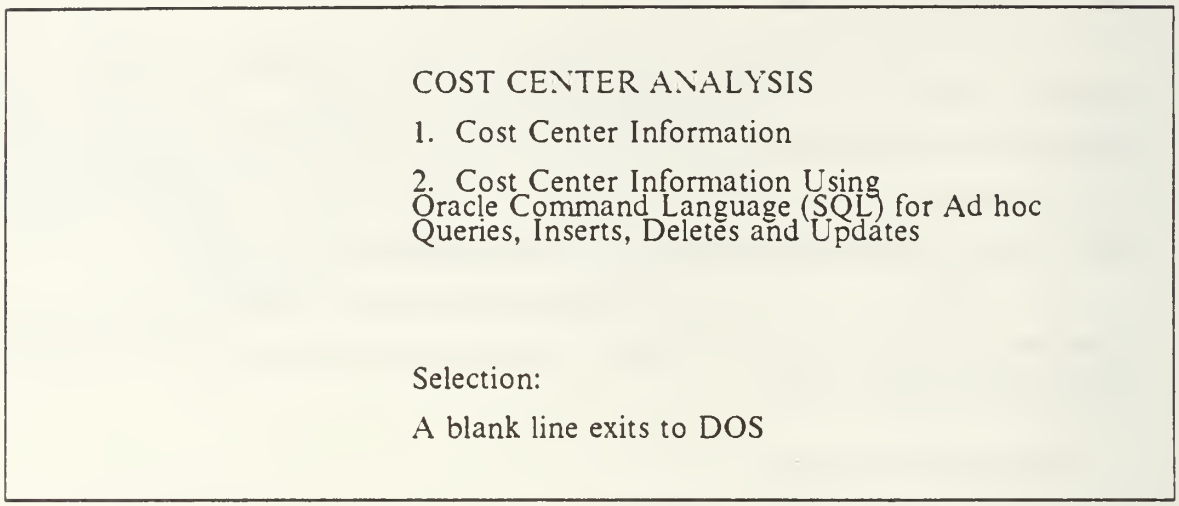


Figure C.1 Cost Center Analysis Main Menu.

At this point you are provided two options (see Figure C.1). If you select option "1," you will be provided menus with preformatted queries. Using these menus simplifies the task of accessing Oracle.

At times the menus may be too limiting or not ask the right questions; therefore, option "2" allows you to use Oracle more directly through the Oracle User Friendly Interface (UFI). To use the UFI, you need to understand the Oracle command language to some degree. In the last section of this appendix we show examples of how to use the command language to develop your own ad hoc queries, make insertions, deletions or drop tables.

After exiting option "2," you will be returned to the A> prompt of DOS. If you wish to reenter the system, it is not necessary to rerun Oracle. You will have to re-enter "CCA" to return to the top menu of the Cost Center Analysis system.

In all menus, when entering options the system will not accept an inappropriate response. Inappropriate values will cycle the user back to the menu he just tried to query from. The system also requires you to verify your responses. If you wish to change your answer, enter "N", or "n" to the question "Is this correct?". The system will blank your response and you can enter your change.

A blank line or a 0 will allow you to exit the present menu and return to the menu directly "above" it.

Note: General instructions for menus will not be repeated under each explanation unless they differ from the norm.

5. INFORMATION AVAILABLE

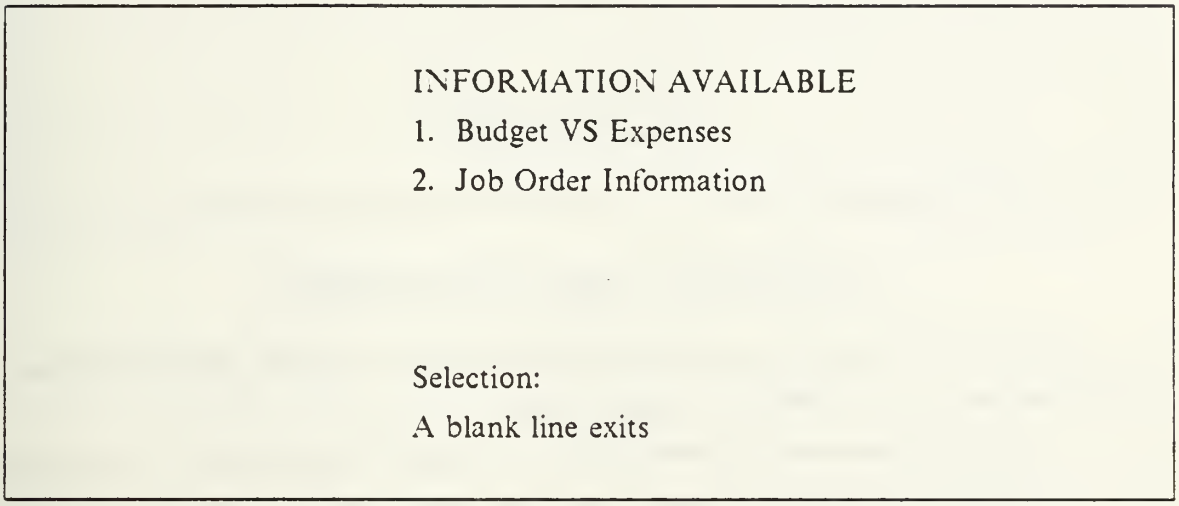


Figure C.2 Information Available Menu.

This menu (see Figure C.2) identifies the classes of information that are available to you.

Option "1" will introduce you to the numerical and computational information available. This is the budget and expense information provided in various formats and aggregations.

Option "2" directly addresses the Job Order information. It will show job orders to budget and cost information.

6. BUDGET VS EXPENSES

This menu allows the user to analyze budget vs expenses under various aggregations (see Figure C.3).

Option "1" provides the user with Total Budget vs Expenses information by Cost Function, Cost Class, Cost Function and Cost Class, or Cost Center.

Option "2" allows the user to closely compare budgeted figures to actual expense figures by either Labor, Material or Other.

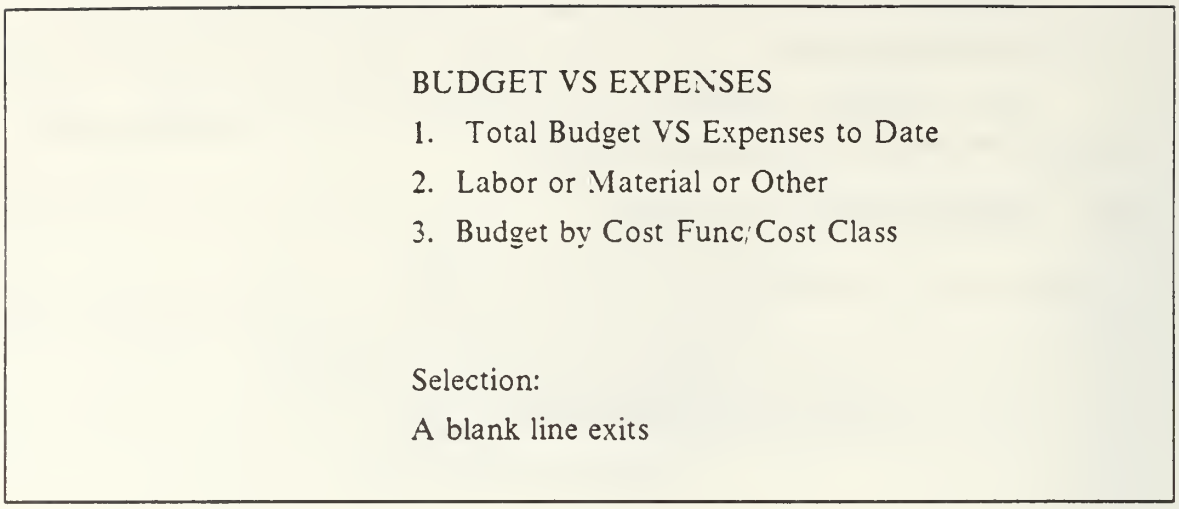


Figure C.3 Budget vs Expenses Menu.

Option "3" gives the user the budget by Cost Function and Cost Class as it would appear on the SBR-22A summary report:

1. A title will appear after you input your selection. Press any key to continue.
2. Select your desired response to the print option prompt. Note: the system will not echo your response, so be patient if it appears to take a little while. The computer must handshake with the printer and this takes a little time.
3. After the screen displays, pressing a key will display the next screen. Pressing a "Q" will abort the rest of the information display for that Cost Function and start the first page of the next cost function's budget.
4. After each Cost Function's budget is displayed you will be asked if you want printed output for the next screen.
5. After the last Cost Function output, you will be returned to the Budget vs Expenses menu.

7. TOTAL BUDGET VS EXPENSES

The first option gives you the total budget vs expenses by Cost Function (see Figure C.4). It allows you to produce a data file for the graphics utilities by answering yes to the graphics output question.

The second option gives you the Total Budget vs Expense by Cost Class. It also can produce the data file for the graphics utility.

The third option provides you the total budget vs expenses information by Cost Function and Cost Class.

The last option outputs the Total Budget vs Expenses by Cost Center. The graphics data file is written if the user so selects.

All expense data is of the most current date entered into the data base.

TOTAL BUDGET VS EXPENSES

1. Cost Function
2. Cost Class
3. Cost Function Cost Class
4. Cost Center

Selection:
A blank line exits

Figure C.4 Total Budget vs Expenses Menu.

CAUTION: Options "1" and "2" write to the same data file. If you select both to create the graphics file during the same session, you will not overwrite but append to the file. The graphics utilities are not designed to accept both types of data in the same file. It is best to have one or the other, but not both types in the file. Option "4" writes two data files for the graphics utilities.

8. BUDGET VS EXPENSES (HOUR, LABOR, MATERIAL OR OTHERS)

BUDGET VS EXPENSES

1. HOURS
2. LABOR
3. MATERIAL
4. OTHER

Selection:
A blank line exits

Figure C.5 Budget vs Expenses by Hour, Labor, Material or Other.

This menu provides a further breakdown of budget vs expenses by Hours, Labor, Material or Other (see Figure C.5).

Option "1" compares budgeted hours to expensed hours by Cost Function/Cost Class.

Option "2" compares budgeted labor to expensed labor by Cost Function/Cost Class.

Option "3" compares budgeted to expensed material by Cost Function/Cost Class.

Option "4" compares budgeted to expensed other by Cost Function/Cost Class.

9. JOB ORDER INFORMATION MENU

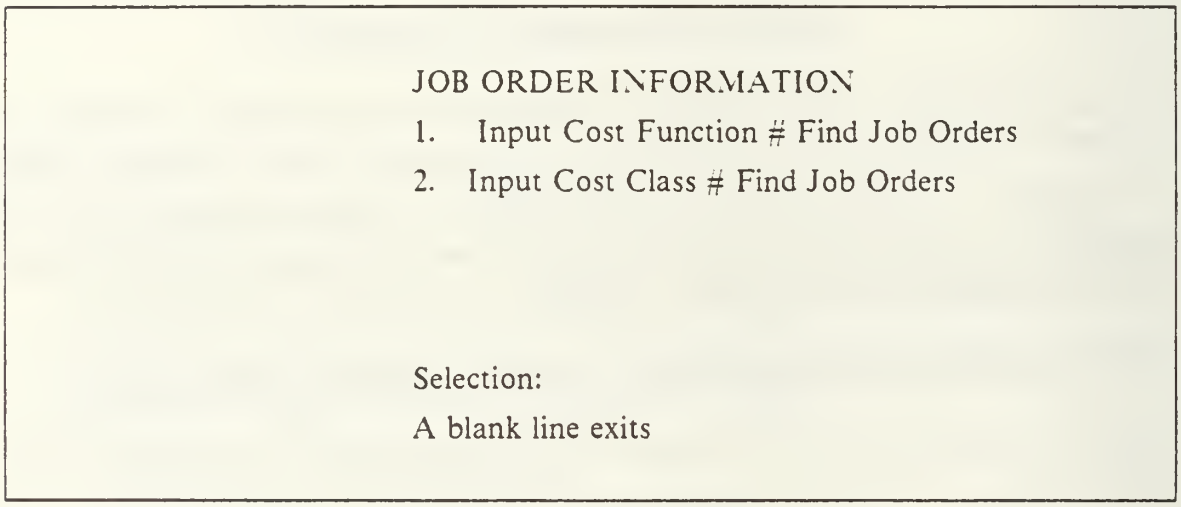


Figure C.6 Job Order Information Menu.

This menu allows the user to select submenus which will list job order numbers associated with a particular Cost Function/Cost Class (see Figure C.6).

Option "1" outputs job orders of the selected Cost Function number. See the Cost Function Input Menu (Figure C.7)

Option "2" outputs job orders of the selected Cost Class numbers. See Cost Class Input Menu (Figure C.8).

10. JOB ORDER NUMBER INPUT MENU

Enter the Cost Function portion first, and press <enter> (see Figure C.9). Then enter the Cost Class portion of the number; press <enter>. Finally, enter the

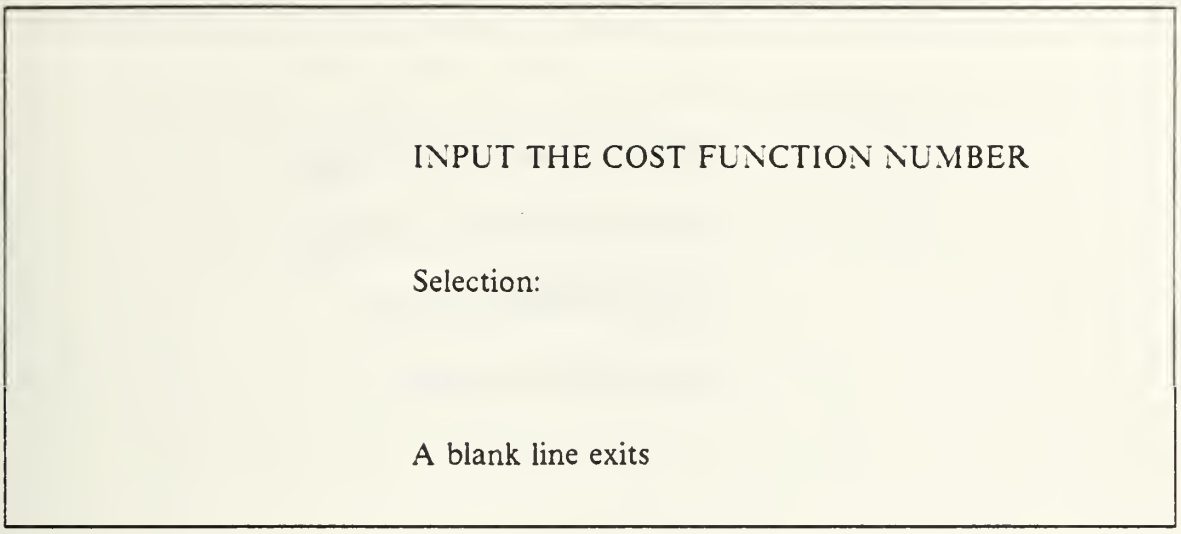


Figure C.7 Cost Function Number Input Menu.

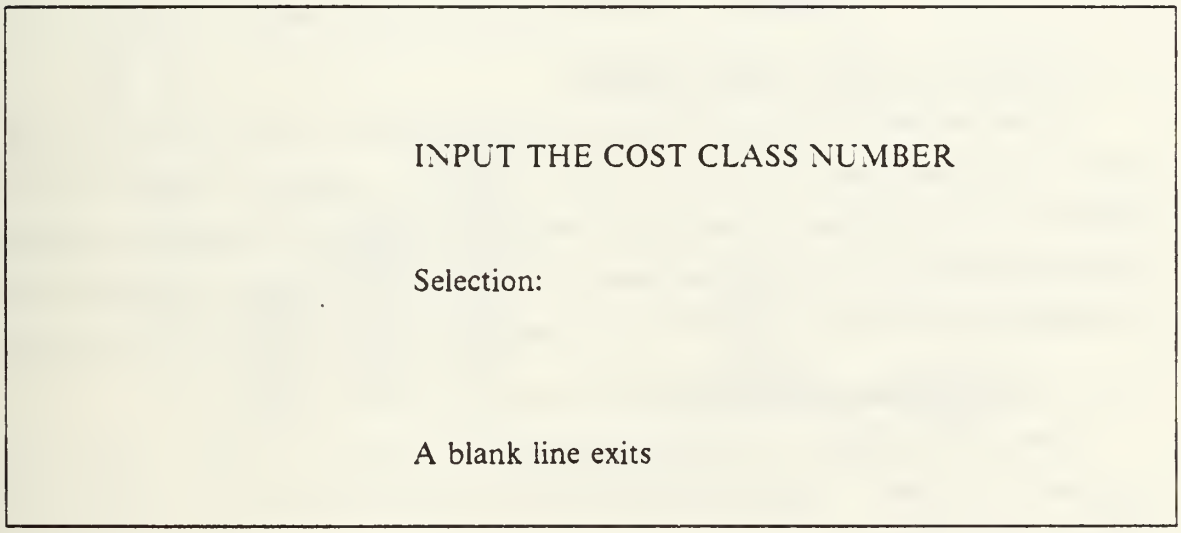


Figure C.8 Cost Class Number Input Menu.

Job Order Number and press <enter>. The system will require you to verify your response. If you wish to make changes, type "N" or "n", and make the changes by reentering all the values. Entering a blanks will return you to the Job Order Information Menu.

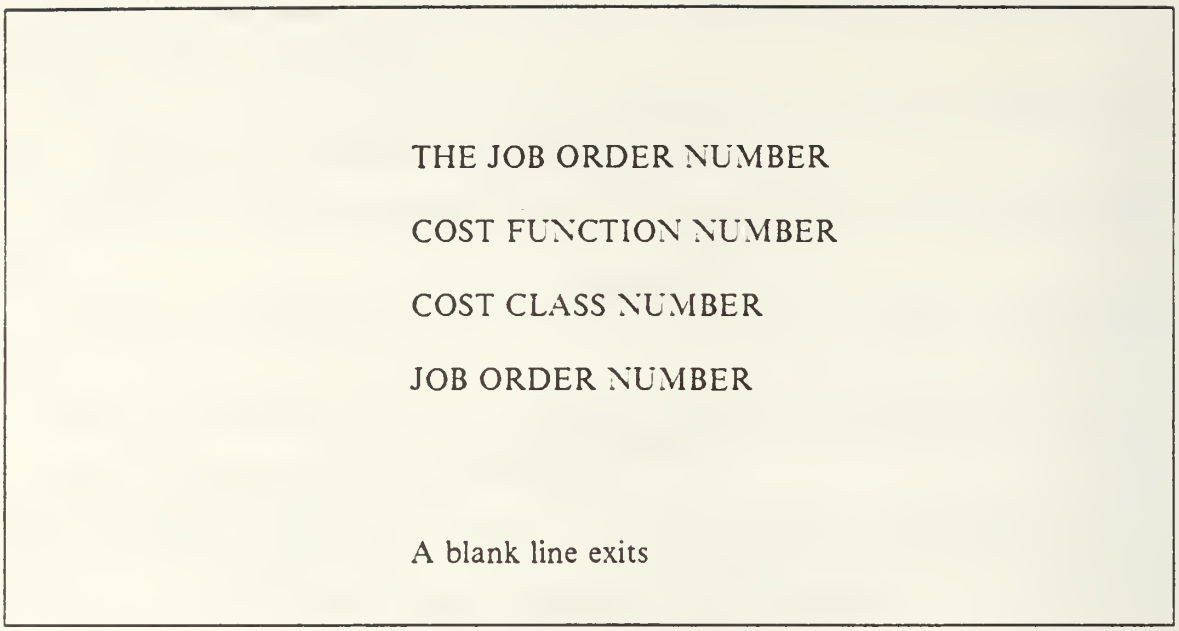


Figure C.9 Job Order Number Input Menu.

11. COST FUNCTION INPUT MENU

Enter the entire four digit Cost Function number, or else no records will be selected by Oracle (see Figure C.7). Press <enter> when the numbers have been inputed. The system will require you to verify your response. Respond to the printed output prompt and then the information of the Cost Function will be displayed. After the information display, type <enter> to return to the Job Order Information menu.

12. COST CLASS INPUT MENU

Enter the two-digit Cost Class number (see Figure C.8). Press <enter> after inputting the number. The system will ask you to verify the response. Respond to the "printed output" prompt and the Job Order numbers for that particular cost class will be displayed. A blank entry returns you to the Job Order Information menu.

13. GRAPHICS

The graphics portion of this system, due to memory constraints, is accessed outside of the system through DOS commands. There are four graphs that the user may view. Each graph is a separate file, so that the user may choose which graph to display. The following is a short description of each graph and the files they access:

BAR.C is executed from DOS by typing "BAR".

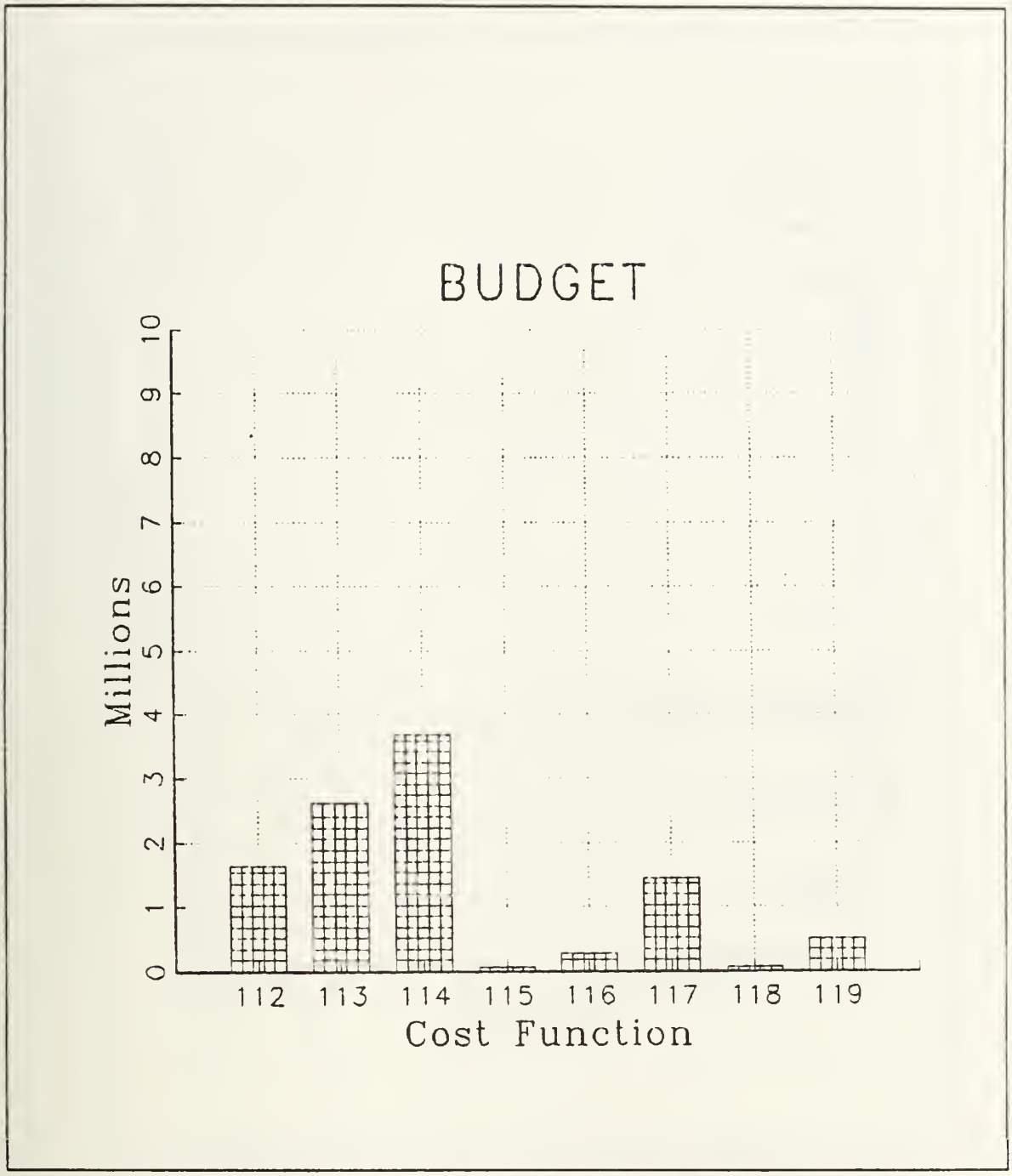


Figure C.10 Single Budget Bar Graph.

This module produces a single bar graph, representing the budget of each Cost Center (see Figure C.10). File accessed: GRAF

PLOT.C is executed from DOS by typing "PLOT".

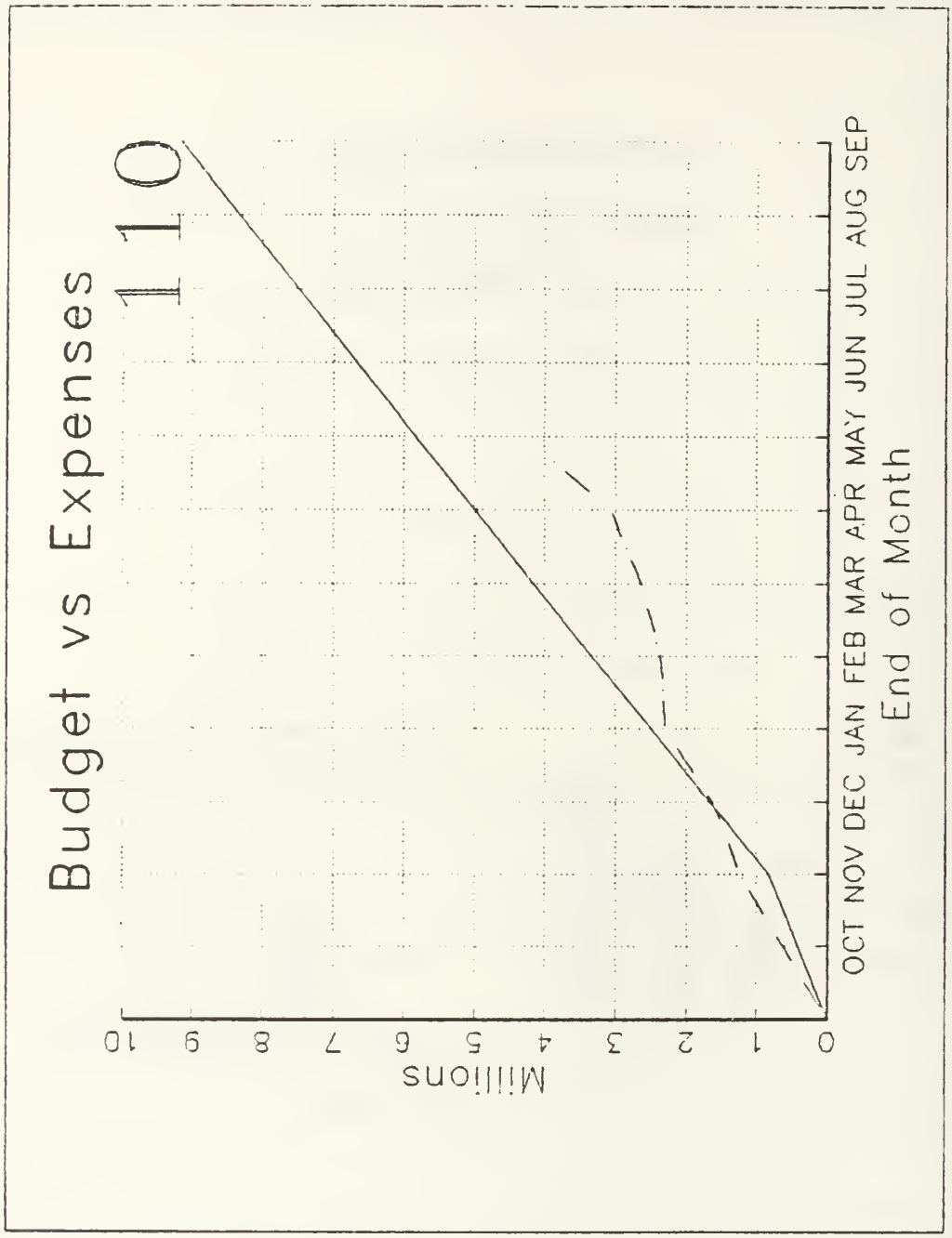


Figure C.11 Plot of Budget and Expenses.

This module produces a line graph. The solid line being the budget, with the broken line representing the expenses. It is plotted by month (see Figure C.11). Files Accessed: GRAF1, BUD

Tripbar.c is executed from DOS by typing "TRIPBAR".

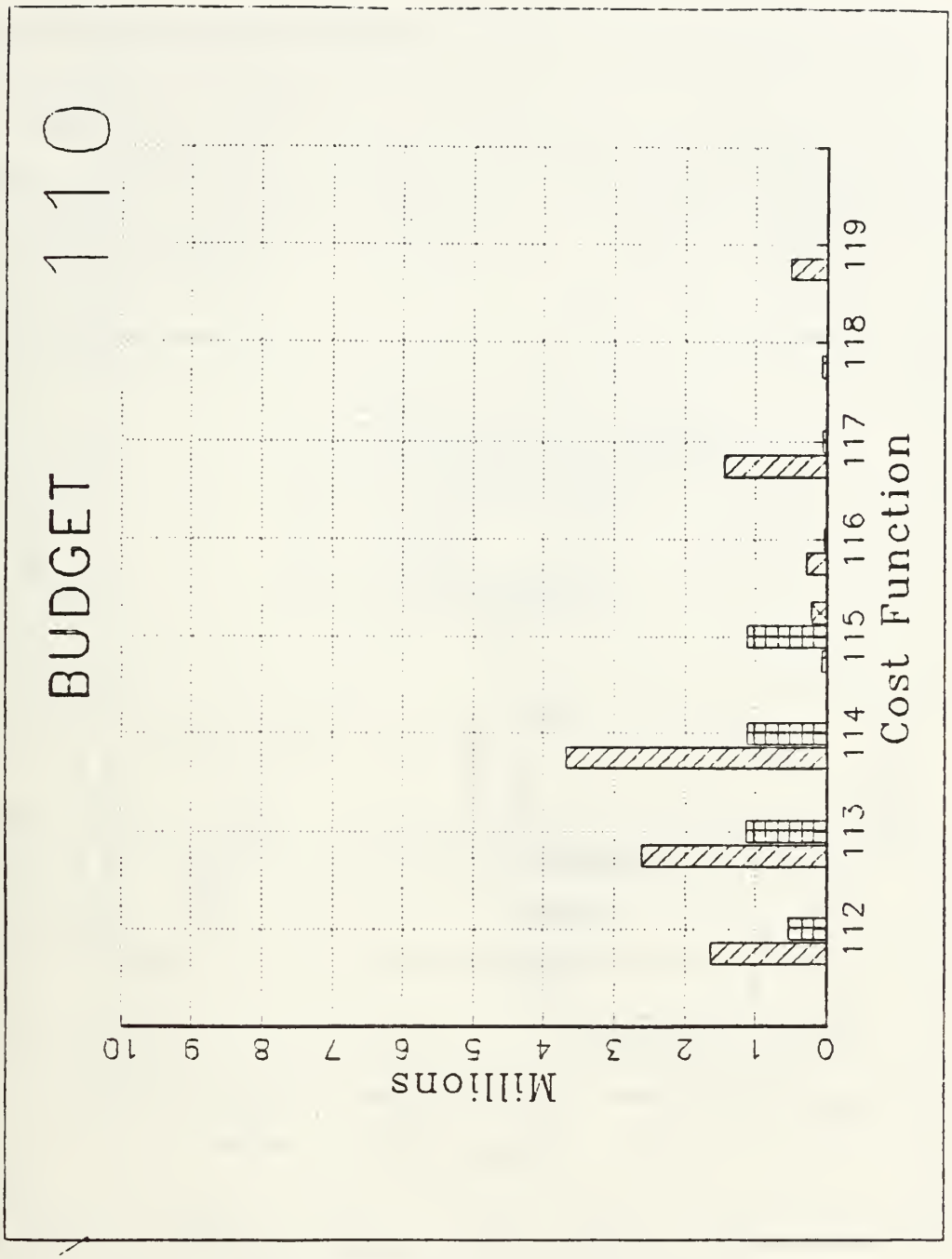


Figure C.12 Triple Bar Graph.

This module produces a Triple bar graph. The middle bar represents the budget for each cost center, left bar represents the expenses and the right bar represents the percentage of the budget expended (see Figure C.12). Files Accessed: GRAF

Combo.c is executed from DOS by typing "COMBO".

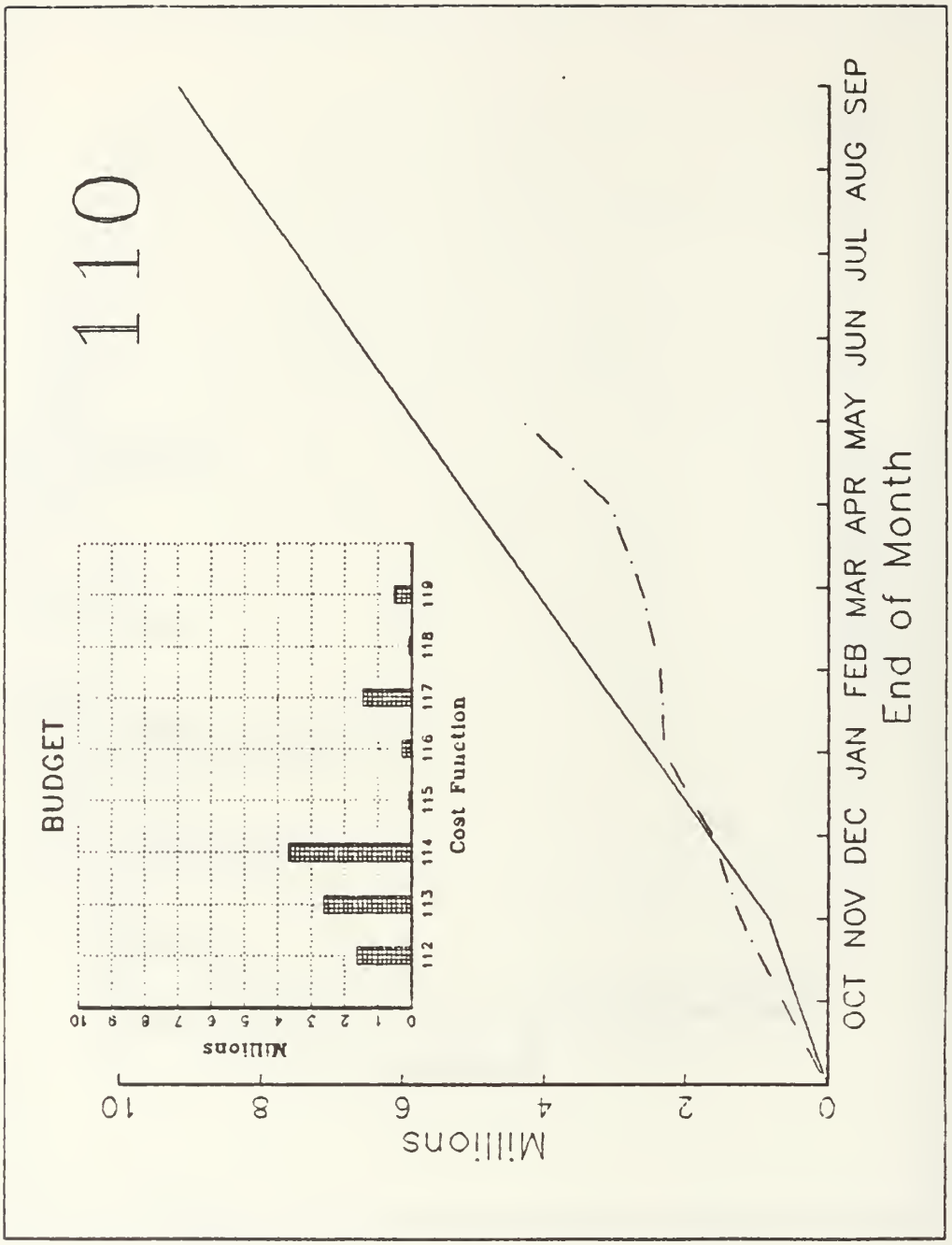


Figure C.13 Plot of Budget and Expenses with Bar Graph Overlayed.

This is called directly from DOS after the CCA system has been processed. This module produces a full page line graph, with the solid line representing the budget and the broken line representing the expenses, by the month. Inset in the upper left hand corner in a single bar graph, reduced in size that plots the budget for each cost center. Files Accessed: GRAF1, BUD, GRAF

If the user is more experienced and plans to use the UFI portion of the system with the graphics modules, he must know the names of the files that each graphic program accesses. Please take note of the above information. After the graph has completed execution, a beep will sound, and the user has a selection of options that he may choose from. The following is a duplication of a menu that will appear if the user presses the <space> bar:

```
l --> large, low resolution plot
L --> large, high resolution plot
m --> medium sized, high resolution plot
M --> medium sized, high resolution plot
g --> draw grid on screen
q --> quit and close files
z --> zoom
w --> zoom with window
1-9 --> ship n pictures
CR --> go on to next plot
```

Figure C.14 Allowable Modifications of Graphs.

See Figure C.14 for the menu of listed options provided the user. With those options, the user has the ability to zoom in for more detail as well as modify the graph produced.

If the user wishes to exit the graph and skip the above menu, he will just press <enter>.

Samples of printed output of all graphs can be found on the following pages.

It should be noted that these programs, while short structurally, take approximately 3 to 4 minutes to complete execution. While the program is displaying the graph on the screen, line by line, it is creating another, faster executing file. If the user executes the program BAR.C by typing "BAR", a corresponding file named BAR.TKF is created. This file is executed by typing "play BAR.TKF".

This file is only a replay of the program BAR.C, and will not contain current data if the data base system has been accessed again. The advantages of having a ".TKF" file is speed. If the user wishes to review plotted data previously created, or if the graphic display is being used in a presentation, there is no need to endure the tedious wait of the main program.

The play option is provided as a utility with the GraphiC graphics library.

14. AD HOC, UPDATES, DELETIONS, MODIFICATIONS WITH ORACLE

a. Introduction

Oracle, through UFI, provides you with many features and procedures. Information can be extracted and displayed in many different ways. The menu driven portion of this system allows you easy access to some of the information provided through this system. However, all queries could not be anticipated and some people prefer to use the command language instead of the menus. For these people we have provided the option of using UFI to interface with Oracle. The UFI interface can also be used to update the data base, delete rows, change attributes and even format reports. We will give you a brief description to get you started. For further information and for more advanced techniques, see the Oracle User Manual Vol. I [Ref. 16] that comes with the Oracle data base.

b. Getting in and out

The first thing you must know is how to get in and out of UFI. You must be in the same directory as the execute file CCA.EXE. Then make sure you run Oracle before running CCA. Type Oracle at the DOS command prompt as shown:

```
D>ORACLE
```

Then type CCA at the prompt.

```
D>CCA
```

When the first menu appears choose 2. If all goes well the next prompt should look like this:

```
UFI>
```

You now should be in UFI. The authorized userid and password were automatically issued in the call to UFI. When you are done and wish to exit type EXIT at the UFI prompt.

```
UFI>EXIT
```

This will return you to the first menu again where you can reenter UFI, use the menu driven queries, or exit to DOS.

c. Ad Hoc Queries

Once you are in UFI and have the UFI prompt, you are ready to begin. First you must know the name of the tables you are dealing with, have an idea of the information stored in each, and the relationships between tables. All this information is contained in the data dictionary (Appendix A). In summary, the following are the names of the tables associated with CCA:

1. EXPENSE

2. BUDGET

The simplest query to make will give you all the information in a particular table. To display this information at the prompt type:

UFI> SELECT * FROM BUDGET; This means select all the columns from the table BUDGET. It will display all the information contained in the table. Be sure to end each query with a semicolon.

Perhaps you do not want all the information in the table but only specific columns. To display this information type at the prompt:

UFI> SELECT COST_FUN_NO, COST_CL_NO, STLABOR, FROM BUDGET;

This will display three columns from the table BUDGET, namely Cost Function number, Cost Class number, and Straight Time Labor for all rows in the table. Notice the commas between the column names. There is no comma between the last column name and the key word FROM. Once again the statement is ended with a semicolon.

If you do no want all the rows but only specific rows, you can limit which ones you get like this:

```
UFI> SELECT * FROM EXPENSE WHERE COST_FUN_NO = '9112';
```

This will display all the information for every record in the EXPENSE table for cost function 9112. The word of number must be exactly as it is in the data base, including capital letters. In UFI, if you forget exactly how the data was entered, Oracle cannot find a match. Again notice the semicolon at the end of the query.

The next step is to combine what we have learned to derive even more specific information. Here is an example:

```
UFI>SELECT COST_FUN_NO, COST_CL_NO, STLABOR FROM
EXPENSE
  2 WHERE COST_FUN_NO = '9118'
  3 AND COST_CL_NO = '54';
```

There are several things to notice on this query. First, it takes more than one line. UFI automatically enters the numbers for each line. Commas are placed only between

column names, but not between STLABOR and the key word FROM. The key word AND must separate the predicates after the where statement. The values that you are looking for must be in the same format as the data is stored, first letter capitalized and the remaining in lower case. The entire query finally must be ended with a semicolon.

Now you can go and look for specific information from a single table. But what if you want information that is contained in two different tables? Do you have to write two different queries comparing the first to the second to find the information you are seeking? No, because this would be the end of this tutorial.

d. **Joins**

Combining tables is known as a join. To join tables they must have a common attribute (column). The name can be different but the values must be stored the same way.

Let's look at an example. Suppose you want to look at budget compared to expenses for a particular date that is in the data base. You want to join BUDGET and EXPENSE. They have two common attributes COST_FUN_NO and COST_CL_NO on which they can be joined. Lets look at this Oracle statement.

```
UFI> SELECT BUDGET.STHOURS, BUDGET.STLABOR,
EXPENSE.STHOURS,
  2 EXPENSE.STLABOR FROM BUDGET, EXPENSE
  3 WHERE BUDGET.COST_FUN_NO = EXPENSE.COST_FUN_NO
  4 AND BUDGET.COST_CL_NO = EXPENSE.COST_CL_NO
  5 AND DT = '31-OCT-86';
```

This will display Straight Time Hours and Straight Time Labor costs for BUDGET and EXPENSE values where the date DT is 31 OCT 86. Notice the attributes that are joined on, COST_FUN_NO and COST_CL_NO are both explicitly stated in the join. The name of the table before the name of the attribute needs to be there only if the names in the tables are the same. Thus the attribute DT in EXPENSE does not have to be written as EXPENSE.DT because there is no DT in the BUDGET table; however, it could be if you prefer for the sake of clarity.

e. **Mathematical Manipulations**

The next topic in Ad Hoc Queries is how to add, subtract, multiply, divide, find the maximum and minimum numbers. We will look at addition. The Other operators work in the same manner.

To add a column of numbers, the command is:

```
UFI> SELECT SUM(STLABOR) FROM BUDGET;
```

or

```
UFI> SELECT SUM(STLABOR), SUM(MATERIAL), SUM(OTHER)
  2  FROM BUDGET;
```

This can also be done with a join:

```
UFI> SELECT SUM(BUDGET.STHOUR), SUM(EXPENSE.STHOUR)
  2  FROM BUDGET, EXPENSE
  3  WHERE BUDGET.COST_FUN_NO = EXPENSE.COST_FUN_NO
  4  AND BUDGET.COST_CL_NO = EXPENSE.COST_CL_NO
  5  AND DT = '31-OCT-86';
```

Rows can also be summed up:

```
UFI> SELECT OTLABOR+STLABOR+MATERIAL+OTHER FROM
BUDGET;
```

Or you can sum both columns and rows at the same time.

```
UFI> SELECT SUM(OTLABOR)+SUM(STLABOR)+SUM(MATERIAL)+
  2  SUM(OTHER) FROM BUDGET;
```

All of the mathematical manipulations are performed in the same way. Consult the Oracle User Manual for more advanced mathematical manipulations.

f. **Group By**

Another useful command is the GROUP BY command. It is especially useful in summarizing information, such as adding columns. Suppose you want the total overtime hours budgeted by each cost function for the entire year. The command to display this information is:

```
UFI> SELECT COST_FUN_NO,SUM(OTHOURS) FROM BUDGET
     GROUP BY COST_FUN_NO;
```

Without the GROUP BY COST_FUN_NO, Oracle would not know how to display the COST_FUN_NO with a sum, since sum returns one total value for all cost functions. With the GROUP BY cost function, Oracle will return a sum for each cost function as shown in Figure C.15

```
COST_FUN_NO    SUM(HOURS)
-----------------------------------------

9112              20
9113             100
9114             450
9115             150
" "              " "
etc. for each cost function in the data base.
```

Figure C.15 Output of Using the GROUP BY Command.

g. **Sub Queries**

A sub query is also useful in summarizing data. A sub query is using another select statement to return a value for the main query. An example might make this a little clearer.

Suppose you want to find the total expenses to date but you do not know the last date of the data in the data base. You could make two separate queries, one to find out what the maximum date is and one to find the sums for that date. Or you could combine it into one query as shown:

```
UFI> SELECT        SUM(OTLABOR+STLABOR+MATERIAL+OTHER)
FROM EXPENSE
     2 WHERE DT = (SELECT MAX(DT) FROM EXPENSE);
```

The sub query returns the maximum value of the date DT which is then used to find the sum. Notice that the rows are added first and then the sum of that resulting column is found.

Sub queries can also return a set of values. Refer to the user manual [Ref. 16] for more on sub queries.

h. **Updates**

As stated previously, the updates will be handled through electronic transfers of data from the Prime Network computer. The manager does have the option of updating on his own, whether to make his data current before the update, or possibly to assist in answering "What if" type questions. The manager could change expense figures, or budget figures to see what effects changes have and then graph out the results.

Updates can also be used to scale values or for global type changes. Or for example, it could be used to change particular values. The following statement multiplies each number in STHOURS of the BUDGET table by 60 to find the number of minutes budgeted for cost function 9118.

```
UFI > UPDATE BUDGET SET STHOURS = HOURS * 60
      2  WHERE COST_FUN_NO = '9118';
```

Update can also be used to change one value:

```
UFI > UPDATE BUDGET SET MATERIAL = 6465
      2  WHERE COST_FUN_NO = '9116'
      3  AND COST_CL_NO = '97';
```

i. **Deletions**

Deletions will generally be taken care of by the updates from the Prime Network, semiweekly. You can make your own deletions if you wish, however.

```
UFI > DELETE FROM BUDGET
      2  WHERE COST_FUN_NO = '9112'
      3  AND COST_CL _NO = '02';
```

This statement would delete the row in the table BUDGET, where Cost Function is 9112 and Cost Class is 02. The same command given on the EXPENSE table would delete a row for every date whose Cost Function is 9112 and whose Cost Class is 02.

j. **Modifications**

Modifications to the existing data base is not recommended on individual bases. Since updates come from the Prime Network, any additions or deletions of columns would make these updates impossible, unless all systems and the Prime Networks reports in which the data is derived are also modified. Under special circumstances a modification may be desirable. To accomplish this the ALTER command is used.

```
UFI> ALTER TABLE BUDGET
     2  ADD (YEAR DATE);
```

The data would then have to be inputted using the UPDATE command described above.

Tables can also be created and destroyed by the CREATE TABLE and the DROP TABLE commands respectively. The easiest way to create a table is to create UFI files as presented in Appendix A. A sample of how to run UFI files is shown below.

1. At the DOS prompt run Oracle.

```
C:ORACLE
```

2. When Dos Prompt returns type UFI followed by the at symbol above the 2, @, and then the UFI filename as shown.

```
C:UFI @BUDGET.UFI
```

3. This will create your table, insert the data and create any indexes as you entered it into the UFI file. See Appendix A for sample UFI files.

The CREATE command is used as shown below:

```
UFI> CREATE TABLE PROJ
     2  (PROJNO NUMBER NOT NULL,
     3  NAME CHAR(10));
```

A different view or subset of a table can also be created in a similar fashion:

```
UFI> CREATE VIEW TELEPHONE AS
```

```
2 SELECT LAST, FIRST, PHONE
3 FROM EMPLOYEE
4 WHERE EID < '400000000';
```

This table would allow you to look at the telephone numbers of employees whose EID is less than 400000000.

The DROP command is used as follows:

```
UFI> DROP TABLE PROJ;
```

The table called PROJ will no longer be in the data base unless recreated.

k. **Other Goodies**

As mentioned earlier, Oracle is a powerful data base management system with a variety of options and commands. Some of the more interesting ones would be ways to present the data in a better format. Although the CCI module in the microcomputer implementation of CCA does much of this for you, you could dress up those answers to queries or even those provided in the CCI.

A few examples of the basic formatting commands include COLUMN, TTITLE, BTITLE, BREAK, and COMPUTE. We shall look mostly at COLUMN and TTITLE here but be aware that these Other commands exist if you need them. COLUMN formats a column's heading and data. Instead of having COST_FUN_NO printed out on a report, you can change it to COST FUNCTION as shown below:

```
UFI> COLUMN COST_FUN_NO HEADING COST FUNCTION

UFI> SELECT COST_FUN_NO, SUM(STLABOR) FROM BUDGET
     2 GROUP BY COST_FUN_NO;
```

TTITLE can then be used to place a title on the page.

```
UFI> TTITLE 'B U D G E T | | BY COST FUNCTION'
```

BTITLE puts a title at the bottom of a page.

BREAK breaks up the report into groups of rows.

COMPUTE computes totals and subtotals on the report.

If you are interested in these commands see the User Manual for Oracle [Ref. 16].

1. **Editing in UFI**

To end this tutorial on the use of UFI, here are some tips on editing. The key commands are CHANGE, LINE, RUN, LIST, INP, AND DEL. To demonstrate these commands let us take a select statement:

```
UFI> SELECT STLABOR, MATERIAL
     2 FROM BUDGET
     3 WHERE COST_FUN_NO = '9118'
     4 AND COST_CL_NO > '91';
```

If you run this and you wish to change something either because of an error or to get a variation on the information, you do not have to type the command in all over.

```
UFI> LIST
```

This will list the last command you typed in. type:

```
UFI> LI
```

That will take you to line 1 and display:

```
1* SELECT STLABOR, MATERIAL
```

```
UFI> CHANGE/MATERIAL/OTHER/
1*  SELECT STLABOR, OTHER
```

This command will change Material to Other in line 1. To execute this type:

```
UFI> RUN
```

This will execute the entire statement. If you want to add lines to the current SQL command use the INP command.

```
UFI > INP
      5  GROUP BY COST_CL_NO
      6
```

This will add this line to the command. Del will delete the current line marked by the *.

```
UFI > LIST
      SELECT LABOR, MATERIAL
      2  FROM BUDGET
      3  WHERE COST_FUN_NO = '9118'
      4  AND COST_CL_NO > '91'
      5* GROUP BY COST_CL_NO;

UFI > DEL
```

Line 5 is now deleted and this SQL statement can be run.

This tutorial briefly describes a procedure that can be followed if you choose to interface with Oracle at the command level. Remember that statements must end with a semicolon and that if all else fails, use the Oracle User Manual [Ref. 16]. The next page summarizes the procedures necessary to access the UfI and key commands to issue UFI and SQL commands.

m. **Summary of UFI and SQL Commands for Command Level Processing**

1. Turn on machine and allow to boot up.
2. Computer boots up onto the A: or C: disk drive depending on the particular configuration.
3. Type in CCA at the appropriate prompt as shown

   C> CCA
4. The first menu gives you a choice of either menu driven or command line. Choose 2 command line.
5. The following is a list of commands and the formats for using them:

```
SELECT ATTRIBUTE1,ATTRIBUTE2,.. FROM TABLENAME
```

```
    WHERE ATTRIBUTE1 = 'VALUE'
    AND ATTRIBUTE2 = (SELECT ATTRIBUTE2 FROM TABLENAME
    WHERE TIME = MAX(TIME);

  UPDATE TABLENAME SET ATTRIBUTE1 = ATTRIBUTE1 / 100
    WHERE ATTRIBUTE1 > 10000;
```

JOINS:

```
  SELECT ATTRIBUTE1, ATTRIBUTE2, ATTRIBUTE3,...
    FROM TABLENAME1, TABLENAME2
    WHERE ATTRIBUTE1 = ATTRIBUTE2;
```

APPENDIX D
CPL AND TELL-A-GRAF PROGRAMS FOR PRIME MINICOMPUTER

1. CPL PROGRAMS

a. PR.CPL

```
/*(PR.CPL) PREPARE REPORTS
/* PROVIDE USER WITH REPORT PRODUCING CAPABILITY WITH TEL-A-GRAF
&ARGS ANS
/*DISPLAY TOP MENU
R DT5
/*GET USER RESPONSE AND VALIDATE
&SET_VAR FLAG := FALSE /* INITIALIZE FLAG TO FALSE
    &DO &UNTIL  %FLAG% = TRUE
        &SET_VAR ANS := [RESPONSE 'Select One']
        &DO CHECK &LIST S T Q
            &IF %ANS% = %CHECK% &THEN &SET_VAR FLAG := TRUE
            &END
        &IF %FLAG% = FALSE &THEN &SET_VAR ANS := [RESPONSE 'Select
One']
        &END
&IF %ANS% = 'T' &THEN R CTEL
&IF %ANS% = 'Q' &THEN &RETURN
&RETURN
```

b. DT5.CPL

```
/*DISPLAY TOP   display top menu
R NL2 27
TYPE '          'T - TEL-A-GRAF GRAPHICS
TYPE '          'Q - QUIT
R NL2 10
&RETURN
```

c. NL2.CPL

```
&ARGS JUMP
&DO M := 0 &TO %JUMP% &BY 1
     TYPE
     &END
&RETURN
```

d. CTEL.CPL

```
/*(CTEL.CPL) CALL TEL-A-GRAF
&DO &UNTIL %FINISH% = TRUE
&SET_VAR ANSWER := [QUERY 'Enter TEL-A-GRAF at COMMAND LEVEL']
&IF %ANSWER% = FALSE &THEN
    R MANTEL
    &ELSE
       R FREE
    &SET_VAR FINISH := [QUERY 'Finished']
    &END
&RETURN
```

e. MANTEL.CPL

```
/*(MANTEL.CPL)  MANIPULATE TEL-A-GRAF
/*Select data, type of graph and open TEL-A-GRAF
&SET_VAR COSTCEN := [RESUME SCC]
&SET_VAR PLOTCODE := [RESUME SPLT]
&SET_VAR PLOTOPT := [RESUME SPLO]
R OPTEL %COSTCEN% %PLOTCODE% %PLOTOPT%
&RETURN
```

f. SCC.CPL

```
/*(SCC.CPL) SELECT COST CENTER
/*Select the desired cost center code for use with TEL-A-GRAF
R DCC
&SET_VAR ANS := [RESUME VCC]
```

```
&RESULT %ANS%
&RETURN
```

g. DCC.CPL

```
/*(DCC.CPL) DISPLAY COST CENTER
/*Display the cost centers that can be graphed with TEL-A-GRAF
R NL2 13
TYPE '                                    'CHOOSE COST CENTER
R NL2 3
TYPE '          '1 - 110
R NL2 10
&RETURN
```

h. VCC.CPL

```
/*(VCC.CPL) VALIDATE COST CENTER CODE
/* Request, get and validate the user response to the Cost Center
menu
&SET_VAR ANS := [RESPONSE 'Select One']
&SET_VAR FLAG := FALSE  /*INITIALIZE FLAG TO FALSE
&DO &UNTIL %FLAG% = TRUE
    &DO CHECK &LIST 1
        &IF %ANS% = %CHECK% &THEN &SET_VAR FLAG := TRUE
        &END
    &IF %FLAG% = FALSE &THEN &SET_VAR ANS := [RESPONSE 'SelectOne']
    &END
&RESULT %ANS%
&RETURN
```

i. SPLT.CPL

```
/*(SPLT.CPL) SELECT PLOT
/*The user is given the ability to select the type of graph TEL-A-GRAF
/*will produce.
R DPLT
```

```
&SET_VAR ANS := [RESUME VPLT]
&RESULT %ANS%
&RETURN
```

j. DPLT.CPL

```
/*(DPLT.CPL) DISPLAY PLOT
/*Display the choices for the type of plot available to the user RNL2
10
TYPE '            'GRAPH PLOT CODE SELECTIONS:
R NL2 4
TYPE '                  'A - PLOT OF EXPENSE TO BUDGET WITH
TYPE '                   '           BARCHART OF BUDGET OVERLAYED ON
THE
R NL2 1
TYPE '                  'B - BARCHART BY COST FUNCTION/COST CLASS OF
TYPE '                  '           EXPENSE TO BUDGET
R NL2 1
TYPE '                  'C - COMPOSITE VARIANCE BARCHARTS
R NL2 10
&RETURN
```

k. VPLT.CPL

```
/*(VPLT.CPL) VALIDATE PLOT CODE
/*Request, get and validate the user response to the Plot Codemenu
&SET_VAR ANS := [RESPONSE 'Select One']
&SET_VAR FLAG := FALSE     /*INITIALIZE FLAG TO FALSE
&DO &UNTIL %FLAG% = TRUE
    &DO CHECK &LIST A B C
        &IF %ANS% = %CHECK% &THEN &SET_VAR FLAG := TRUE
        &END
    &IF %FLAG% = FALSE &THEN &SET_VAR ANS := -RESPONSE 'Select One']
    &END
&RESULT %ANS%
```

```
&RETURN
```

l. SPLO.CPL

```
/*(SPLO.CPL) SELECT PLOT OPTIONS
/*Select the option to plot the total cost center or a cost function
/* within it .
R DPLO
&SET_VAR ANS := [RESUME VPLO]
&RESULT %ANS%
&RETURN
```

m. DPLO.CPL

```
/*(DPLO.CPL) DISPLAY PLOT OPTIONS
/*Display the menu cost functions that can be plotted under the
/* cost center and for the plot type selected
R NL2 10
TYPE '          'PLOT OPTIONS:
R NL2 2
TYPE '                          'A - TOTAL
TYPE '                          '2 - 112
TYPE '                          '3 - 113
TYPE '                          '4 - 114
TYPE '                          '5 - 115
TYPE '                          '6 - 116
TYPE '                          '7 - 117
TYPE '                          '8 - 118
TYPE '                          '9 - 119
R NL2 10
&RETURN
```

n. VPLO.CPL

```
/*(VPLO.CPL)  VALIDATE PLOT OPTIONS CODE
/*Request, get and validate the user response to the Plot Options menu
&SET_VAR ANS := [RESPONSE 'Select One']
&SET_VAR FLAG := FALSE  /*INITIALIZE FLAG TO FALSE
&DO &UNTIL %FLAG% = TRUE
    &DO CHECK &LIST A 2 3 4 5 6 7 8 9
        &IF %ANS% = %CHECK% &THEN &SET_VAR FLAG := TRUE
        &END
    &IF %FLAG% = FALSE &THEN &SET_VAR ANS := [RESPONSE 'Select One']
    &END
&RESULT %ANS%
&RETURN
```

o. OPTEL.CPL

```
/*(OPTEL.CPL) OPEN TEL-A-GRAF
/*Open TEL-A-GRAF and input the user's graph selection.  If Free
/*Form is selected the user will input the graph selections.  &ARGS
COSTCEN; PLOTCODE; PLOTOPT
&SET_VAR SECONDATA := ''
&SET_VAR THIRDINCLUDE := ''
&SET_VAR FOURTHINCLUDE := ''
&IF %COSTCEN%%PLOTCODE%%PLOTOPT% = 1AA &THEN
    &DO
        &SET_VAR DATAFILE := "BE110"
        &SET_VAR SECONDATA := "B110"
        &SET_VAR INCLUDEFILE := "EX2"
        &SET_VAR SECONDINCLUDE := "B1"
        &END
&IF %COSTCEN%%PLOTCODE%%PLOTOPT% = 1A2 &THEN
    &DO
        &SET_VAR DATAFILE := "BE112"
        &SET_VAR SECONDATA := "B110"
        &SET_VAR INCLUDEFILE := "EX2"
```

```
        &SET_VAR SECONDINCLUDE := "EX112"
        &SET_VAR THIRDINCLUDE := "B1"
        &SET_VAR FOURTHINCLUDE := ''
        &END
&IF %COSTCEN%%PLOTCODE%%PLOTOPT% = 1A3 &THEN
    &DO
        &SET_VAR DATAFILE := "BE113"
        &SET_VAR SECONDATA .= "B110"
        &SET_VAR INCLUDEFILE := "EX2"
        &SET_VAR SECONDINCLUDE := "EX113"
        &SET_VAR THIRDINCLUDE := "B1"
        &SET_VAR FOURTHINCLUDE := ''
        &END
&IF %COSTCEN%%PLOTCODE%%PLOTOPT% = 1A4 &THEN
    &DO
        &SET_VAR DATAFILE := "BE114"
        &SET_VAR SECONDATA := "B110"
        &SET_VAR INCLUDEFILE := "EX2"
        &SET_VAR SECONDINCLUDE := "EX114"
        &SET_VAR THIRDINCLUDE := "B1"
        &SET_VAR FOURTHINCLUDE := ''
        &END
&IF %COSTCEN%%PLOTCODE%%PLOTOPT% = 1A5 &THEN
    &DO
        &SET_VAR DATAFILE := "BE115"
        &SET_VAR SECONDATA := "B11O"
        &SET_VAR INCLUDEFILE := "EX2"
        &SET_VAR SECONDINCLUDE := "EX115"
        &SET_VAR THIRDINCLUDE := "B1"
        &SET_VAR FOURTHINCLUDE := ''
        &END
&IF %COSTCEN%%PLOTCODE%%PLOTOPT% = 1A6 &THEN
    &DO
        &SET_VAR DATAFILE := "BE116"
        &SET_VAR SECONDATA := "B110"
```

```
        &SET_VAR INCLUDEFILE := "EX2"
        &SET_VAR SECONDINCLUDE := "EX116"
        &SET_VAR THIRDINCLUDE := "B1"
        &SET_VAR FOURTHINCLUDE := ''
        &END
&IF %COSTCEN%%PLOTCODE%%PLOTOPT% = 1A7 &THEN
    &DO
        &SET_VAR DATAFILE := "BE117"
        &SET_VAR SECONDATA := "B110"
        &SET_VAR INCLUDEFILE := "EX2"
        &SET_VAR SECONDINCLUDE := "EX117"
        &SET_VAR THIRDINCLUDE := "B1"
        &SET_VAR FOURTHINCLUDE := ''
        &END
&IF %COSTCEN%%PLOTCODE%%PLOTOPT% = 1A8 &THEN
    &DO
        &SET_VAR DATAFILE := "BE118"
        &SET_VAR SECONDATA := "B110"
        &SET_VAR INCLUDEFILE := "EX2"
        &SET_VAR SECONDINCLUDE := "EX118"
        &SET_VAR THIRDINCLUDE := "B1"
        &SET_VAR FOURTHINCLUDE := ''
        &END
&IF %COSTCEN%%PLOTCODE%%PLOTOPT% = 1A9 &THEN
    &DO
        &SET_VAR DATAFILE := "BE119"
        &SET_VAR SECONDATA := "B110"
        &SET_VAR INCLUDEFILE := "EX2"
        &SET_VAR SECONDINCLUDE := "EX119"
        &SET_VAR THIRDINCLUDE := "B1"
        &SET_VAR FOURTHINCLUDE := ''
        &END
&IF %COSTCEN%%PLOTCODE%%PLOTOPT% = 1BA &THEN
    &DO
        &SET_VAR DATAFILE := "BBE110"
```

```
        &SET_VAR INCLUDEFILE := "B4"
        &SET_VAR SECONDINCLUDE := ''
        &END
&IF %COSTCEN%%PLOTCODE%%PLOTOPT% = 1B2 &THEN
    &DO
        &SET_VAR DATAFILE := "BBE112"
        &SET_VAR INCLUDEFILE := "B4"
        &SET_VAR SECONDINCLUDE := "B112"
        &END
&IF %COSTCEN%%PLOTCODE%%PLOTOPT% = 1B3 &THEN
    &DO
        &SET_VAR DATAFILE := "BBE113"
        &SET_VAR INCLUDEFILE := "B4"
        &SET_VAR SECONDINCLUDE := "B113"
        &END
&IF %COSTCEN%%PLOTCODE%%PLOTOPT% = 1B4 &THEN
    &DO
        &SET_VAR DATAFILE := "BBE114"
        &SET_VAR INCLUDEFILE := "B4"
        &SET_VAR SECONDINCLUDE := "B114"
        &END
&IF %COSTCEN%%PLOTCODE%%PLOTOPT% = 1B5 &THEN
    &DO
        &SET_VAR DATAFILE := "BBE115"
        &SET_VAR INCLUDEFILE := "B4"
        &SET_VAR SECONDINCLUDE := "B115"
        &END
&IF %COSTCEN%%PLOTCODE%%PLOTOPT% = 1B6 &THEN
    &DO
        &SET_VAR DATAFILE := "BBE116"
        &SET_VAR INCLUDEFILE := "B4"
        &SET_VAR SECONDINCLUDE := "B116"
        &END
&IF %COSTCEN%%PLOTCODE%%PLOTOPT% = 1B7 &THEN
    &DO
```

```
        &SET_VAR DATAFILE := "BBE117"
        &SET_VAR INCLUDEFILE := "B4"
        &SET_VAR SECONDINCLUDE := "B117"
        &END
&IF %COSTCEN%%PLOTCODE%%PLOTOPT% = 1B8 &THEN
    &DO
        &SET_VAR DATAFILE := "BBE118"
        &SET_VAR INCLUDEFILE := "B4"
        &SET_VAR SECONDINCLUDE := "B118"
        &END
&IF %COSTCEN%%PLOTCODE%%PLOTOPT% = 1B9 &THEN
    &DO
        &SET_VAR DATAFILE := "BBE119"
        &SET_VAR INCLUDEFILE := "B4"
        &SET_VAR SECONDINCLUDE := "B119"
        &END
&IF %COSTCEN%%PLOTCODE%%PLOTOPT% = 1CA &THEN
    &DO
        &SET_VAR DATAFILE := "PB110"
        &SET_VAR INCLUDEFILE := "PERBAR"
        &SET_VAR MESSAGE := "9110"
        &SET_VAR SECONDATA := "NB110"
        &SET_VAR SECONDINCLUDE := "NORBAR"
        &SET_VAR THIRDATA := "VB110"
        &SET_VAR THIRDINCLUDE := "VARBAR"
        &SET_VAR FOURTHDATA := "PV110"
        &SET_VAR FOURTHINCLUDE := "PERVAR"
        &END
&IF %COSTCEN%%PLOTCODE%%PLOTOPT% = 1C2 &THEN
    &DO
        &SET_VAR DATAFILE := "PB112"
        &SET_VAR INCLUDEFILE := "PERBAR"
        &SET_VAR SECONDATA := "NB112"
        &SET_VAR SECONDINCLUDE := "NORBAR"
        &SET_VAR THIRDATA := "VB112"
```

```
        &SET_VAR THIRDINCLUDE := "VARBAR"
        &SET_VAR FOURTHDATA := "PV112"
        &SET_VAR FOURTHINCLUDE := "PERVAR"
        &SET_VAR MESSAGE := "9112"
        &END
&IF %COSTCEN%%PLOTCODE%%PLOTOPT% = 1C3 &THEN
    &DO
        &SET_VAR DATAFILE := "PB113"
        &SET_VAR INCLUDEFILE := "PERBAR"
        &SET_VAR SECONDATA := "NB113"
        &SET_VAR SECONDINCLUDE := "NORBAR"
        &SET_VAR THIRDATA := "VB113"
        &SET_VAR THIRDINCLUDE := "VARBAR"
        &SET_VAR FOURTHDATA := "PV113"
        &SET_VAR FOURTHINCLUDE := "PERVAR"
        &SET_VAR MESSAGE := "9113"
        &END
&IF %COSTCEN%%PLOTCODE%%PLOTOPT% = 1C4 &THEN
    &DO
        &SET_VAR DATAFILE := "PB114"
        &SET_VAR INCLUDEFILE := "PERBAR"
        &SET_VAR SECONDATA := "NB114"
        &SET_VAR SECONDINCLUDE := "NORBAR"
        &SET_VAR THIRDATA := "VB114"
        &SET_VAR THIRDINCLUDE := "VARBAR"
        &SET_VAR FOURTHDATA := "PV114"
        &SET_VAR FOURTHINCLUDE := "PERVAR"
        &SET_VAR MESSAGE := "9114"
        &END
&IF %COSTCEN%%PLOTCODE%%PLOTOPT% = 1C5 &THEN
    &DO
        &SET_VAR DATAFILE := "PB115"
        &SET_VAR INCLUDEFILE := "PERBAR"
        &SET_VAR SECONDATA := "NB115"
        &SET_VAR SECONDINCLUDE := "NORBAR"
```

```
        &SET_VAR THIRDATA := "VB115"
       &SET_VAR THIRDINCLUDE := "VARBAR"
       &SET_VAR FOURTHDATA := "PV115"
       &SET_VAR FOURTHINCLUDE := "PERVAR"
       &SET_VAR MESSAGE := "9115"
       &END
&IF %COSTCEN%%PLOTCODE%%PLOTOPT% = 1C6 &THEN
    &DO
        &SET_VAR DATAFILE := "PB116"
        &SET_VAR INCLUDEFILE := "PERBAR"
        &SET_VAR SECONDATA := "NB116"
        &SET_VAR SECONDINCLUDE := "NORBAR"
        &SET_VAR THIRDATA := "VB116"
        &SET_VAR THIRDINCLUDE := "VERBAR"
        &SET_VAR FOURTHDATA := "PV116"
        &SET_VAR FOURTHINCLUDE := "PERVAR"
        &SET_VAR MESSAGE := "9116"
        &END
&IF %COSTCEN%%PLOTCODE%%PLOTOPT% = 1C7 &THEN
    &DO
        &SET_VAR DATAFILE := "PB117"
        &SET_VAR INCLUDEFILE := "PERBAR"
        &SET_VAR SECONDATA := "NB117"
        &SET_VAR SECONDINCLUDE := "NORBAR"
        &SET_VAR THIRDATA := "VB117"
        &SET_VAR THIRDINCLUDE := "VERBAR"
        &SET_VAR FOURTHDATA := "PV117"
        &SET_VAR FOURTHINCLUDE := "PERVAR"
        &SET_VAR MESSAGE := "9117"
        &END
&IF %COSTCEN%%PLOTCODE%%PLOTOPT% = 1C8 &THEN
    &DO
        &SET_VAR DATAFILE := "PB118"
        &SET_VAR INCLUDEFILE := "PERBAR"
        &SET_VAR SECONDATA := "NB118"
```

```
        &SET_VAR SECONDINCLUDE := "NORBAR"
        &SET_VAR THIRDATA := "VB118"
        &SET_VAR THIRDINCLUDE := "VERBAR"
        &SET_VAR FOURTHDATA := "PV118"
        &SET_VAR FOURTHINCLUDE := "PERVAR"
        &SET_VAR MESSAGE := "9118"
        &END
 &IF %COSTCEN%%PLOTCODE%%PLOTOPT% = 1C9 &THEN
    &DO
        &SET_VAR DATAFILE := "PB119"
        &SET_VAR INCLUDEFILE := "PERBAR"
        &SET_VAR SECONDATA := "NB119"
        &SET_VAR SECONDINCLUDE := "NORBAR"
        &SET_VAR THIRDATA := "VB119"
        &SET_VAR THIRDINCLUDE := "VERBAR"
        &SET_VAR FOURTHDATA := "PV119"
        &SET_VAR FOURTHINCLUDE := "PERVAR"
        &SET_VAR MESSAGE := "9119"
 &IF %SECONDINCLUDE% = '' &THEN R SINGLE %DATAFILE% %INCLUDEFILE%
    &ELSE &IF %SECONDATA% = '' &THEN
          R DOUBAR %DATAFILE% %INCLUDEFILE% %SECONDINCLUDE%
        &ELSE &IF %THIRDINCLUDE% = '' &THEN
           R DOUBLE %DATAFILE% %SECONDATA% %INCLUDE% %SECONDINCLUDE%
 %SECONDINCLUDE%
           &ELSE &IF %FOURTHINCLUDE% = '' &THEN
              R    TRIPLE    %DATAFILE%    %SECONDATA%    %INCLUDEFILE%
%SECONDINCLUDE% %THIRDINCLUDE%
              &ELSE   R   QUAD   %DATAFILE%   %SECONDATA%   %THIRDATA%
%FOURTHDATA%
                  %INCLUDEFILE%     %SECONDINCLUDE%     %THIRDINCLUDE%
%FOURTHINCLUDE% %MESSAGE%
 &RETURN
```

p. FREE.CPL

```
/*(FREE.CPL) FREE FORM INPUT TO TEL-A-GRAF
/*Allow the experienced user to manipulate TEL-A-GRAF using it's
commands
5.8 TAG
&TTY
&END
&RETURN
```

q. SINGLE.CPL

```
/*(SINGLE.CPL) SINGLE INCLUDE FILE
/*Allows user to input a datafile and include file to TEL-A-GRAF,
having
/*little or no knowledge of TEL-A-GRAF commands.
&ARGS DATAFILE; INCLUDEFILE
5.8 TAG
DATA FILE IS %DATAFILE%.
INCLUDE %INCLUDEFILE%.
SUBPLOT 1.
DRAW 1.
&END
&RETURN
```

r. DOUBLE.CPL

```
/*(DOUBLE.CPL) DOUBLE DATA AND INCLUDE FILES /*Allows the user to
input two include files to TEL-A-GRAF with little or no
/*knowledge or experience with TEL-A-GRAF commands.
&ARGS DATAFILE; SECONDATA; INCLUDEFILE; SECONDINCLUDE
5.8 TAG
DATA FILE IS %DATAFILE%.
INCLUDE %INCLUDEFILE%.
SUBPLOT 1.
DATAFILE IS %SECONDATA%.
INCLUDE %SECONDINCLUDE%.
```

```
SUBPLOT 2.
DRAW 1 2.
&TTY
&END
&RETURN
```

s. DOUBAR.CPL

```
/*(DOUBAR.CPL) DOUBLE INCLUDE FILES
/*Allows the user to input two include files to TEL-A-GRAF with
little or no
/*knowledge or experience with TEL-A-GRAF commands.
&ARGS DATAFILE; INCLUDEFILE; SECONDINCLUDE 5.8 TAG
DATA FILE IS %DATAFILE%.
INCLUDE %INCLUDEFILE%.
INCLUDE %SECONDINCLUDE%.
SUBPLOT 1.
DRAW 1.
&TTY
&END
&RETURN
```

t. TRIPLE.CPL

```
/*(TRIPLE.CPL) DOUBLE DATA AND TRIPLE INCLUDE FILES /*Allows the
user to input three include files to TEL-A-GRAF with little or no
/*knowledge or experience with TEL-A-GRAF commands.
&ARGS DATAFILE; SECONDATA; INCLUDEFILE; SECONDINCLUDE; THIRDINCLUDE
5.8 TAG
DATA FILE IS %DATAFILE%.
INCLUDE %INCLUDEFILE%.
SUBPLOT 1.
DATAFILE IS %SECONDATA%.
INCLUDE %SECONDINCLUDE%.
INCLUDE %THIRDINCLUDE%.
SUBPLOT 2.
```

```
DRAW 1 2.
&TTY
&END
&RETURN
```

u. QUAD.CPL

```
/*(QUAD.CPL) QUAD DATA AND INCLUDE FILES
/*Allows the user to input four include files to TEL-A-GRAF with
little or no
/*knowledge or experience with TEL-A-GRAF commands.
&ARGS   DATAFILE;   SECONDATA;   THIRDATA;   FOURTHDATA;   INCLUDEFILE;
SECONDINCLUDE; THIRDINCLUDE; FOURTHINCLUDE; MESSAGE
5.8 TAG
DATA FILE IS %DATAFILE%.
INCLUDE %INCLUDEFILE%.
C.
TITLE TEXT IS "PERCENT EXPENDED %MESSAGE%.
SUBPLOT 1.
DATA FILE IS %SECONDATA%.
INCLUDE %SECONDINCLUDE%.
DATA FILE IS %THIRDATA%.
INCLUDE %THIRDINCLUDE%.
DATA FILE IS %FOURTHDATA%.
INCLUDE %FOURTHINCLUDE%.
DRAW 1 2 3 4.
&TTY
&END
&RETURN
```

2. TELL-A-GRAF PROGRAMS

a. TAGPRO.DAT: Tell-A-Graf Profile File

```
PRIMARY DEVICE IS TEKTRONIX.
PRIMARY DEVICE MODEL IS 4105.
PRIMARY DEVICE DRAWING ORDER IS 0.
SECONDARY DEVICE IS POP.
```

```
PAGE LAYOUT IS HORIZONTAL-REPORT.
ERROR REPORTING LEVEL IS 3.
EXIT.
```

b. B1: Bar Chart For Budget

```
GENERATE A VERTICAL BAR CHART .
INDEPENDENT DIVISION-LABELS IS '112' '113' '114'
        '115' '116'  '117'  '118' '119' .
INDEPENDENT LABEL TEXT IS "COST FUNCTION" .
DEPENDENT LABEL TEXT IS "MILLIONS OF DOLLARS" .
AXIS FRAME IS 1.
WINDOW DESTINATION IS  -1 6 4.099999 10.
WINDOW DESTINATION FRAME IS  0.
TITLE TEXT IS "FY 86 BUDGET FOR" "COST CENTER 9110" .
**FILE**
```

c. EX2: Plot of Budget vs Expense

```
GENERATE A PLOT .
X AXIS DIVISION-LABELS IS "OCT" "NOV" "DEC" "JAN"
        "FEB" "MAR" "APR" "MAY" "JUN" "JUL" "AUG" "SEP" .
X AXIS GRID IS 0.
X AXIS LENGTH IS  8.5.
X AXIS LABEL TEXT IS "END OF MONTH" . X AXIS SHIFT IS 1.
Y AXIS GRID IS  0.
Y AXIS MODE IS REVERSED.
Y AXIS OFFSET IS  8.
AXIS FRAME IS  0.
WINDOW DESTINATION IS -1 10 -2 10.
LEGEND FRAME IS 1.
LEGEND X ORIGIN IS  11.
LEGEND Y ORIGIN IS  2.
LEGEND UNITS IS COORDINATE.
MESSAGE 1.
MESSAGE CONNECT POINT IS 0.5 -0.5.  MESSAGE TEXT IS
"BUDGET VS EXPENSES" "COST CENTER 9110"
        "FOR ALL COST FUNCTIONS" .
```

```
MESSAGE UNITS IS INCHES.
MESSAGE X IS  9.
MESSAGE Y IS  6.
MESSAGE 2.
MESSAGE CONNECT POINT IS 0.5 -0.5.
MESSAGE TEXT IS "MILLIONS" .
MESSAGE UNITS IS COORDINATE.
MESSAGE X IS  10.5.
MESSAGE Y IS  4.
**FILE**
```

d. EX112: File Appended to EX2 For 9112

```
MESSAGE 1 "BUDGET VS EXPENSES"  "FOR COST FUNCTION 9112".
MESSAGE 2 TEXT IS "THOUSANDS".
**FILE**
```

e. EX113: File Appended to EX2 for 9113

```
MESSAGE 1 "BUDGET VS EXPENSES"  "FOR COST FUNCTION 9113".
MESSAGE 2 IS "THOUSANDS".
**FILE**
```

f. EX114: File Appended to EX2 for 9114

```
MESSAGE 1 "BUDGET VS EXPENSES"  "FOR COST FUNCTION 9114".
MESSAGE 2 IS "THOUSANDS".
**FILE**
```

g. EX115: File Appended to EX2 for 9115

```
MESSAGE 1 "BUDGET VS EXPENSES"  "FOR COST FUNCTION 9115".
MESSAGE 2 IS "THOUSANDS".
**FILE**
```

h. EX116: File Appended to EX2 for 9116

```
MESSAGE 1 "BUDGET VS EXPENSES"  "FOR COST FUNCTION 9116".
MESSAGE 2 IS "THOUSANDS".
**FILE**
```

i. EX117: File Appended to EX2 for 9117

```
MESSAGE 1 "BUDGET VS EXPENSES"  "FOR COST FUNCTION 9117".
MESSAGE 2 IS "THOUSANDS".
**FILE**
```

j. EX118: File Appended to EX2 for 9118

```
MESSAGE 1 "BUDGET VS EXPENSES"  "FOR COST FUNCTION 9118".
MESSAGE 2 IS "THOUSANDS".
**FILE**
```

k. EX119: File Appended to Ex2 for 9119

```
MESSAGE 1 "BUDGET VS EXPENSES"  "FOR COST FUNCTION 9119".
MESSAGE 2 IS "THOUSANDS".
**FILE**
```

l. B4: Triple Bar Chart, Budget, Budget %, Expense

```
GENERATE A VERTICAL BAR CHART .
INDEPENDENT DIVISION-LABELS IS '112' '113' '114'
        '115'  '116' '117' '118' '119' .
INDEPENDENT LABEL TEXT IS "COST FUNCTION" .
DEPENDENT GRID IS 1.
DEPENDENT LABEL TEXT IS "MILLIONS OF DOLLARS" .
AXIS FRAME IS 0.
TITLE TEXT IS "FY 86 BUDGET VS EXPENSES"
                  "COST CENTER 110".
LEGEND FRAME IS  1.
**FILE**
```

m. B112: Appends B4 for 9112

```
DIVISION LABELS '02'  '03' '04' '11'  '12'  '19'
        '30' '33' '39' '68' '91' '93' '96' '97' '98'.
X AXIS LABEL IS "COST CLASS".
TITLE IS "FY 86 BUDGET VS EXPENSES" "COST FUNCTION 9112".
DEPENDENT LABEL TEXT IS "THOUSANDS OF DOLLARS".
**FILE**
```

n. B113: Appends B4 for 9113

```
DIVISION LABELS '04' '39' '91' '92' '93' '96' '99'.
X AXIS LABEL IS "COST CLASS".
TITLE IS "FY 86 BUDGET VS EXPENSES"
        "COST FUNCTION 9113".
DEPENDENT LABEL TEXT IS "THOUSANDS OF DOLLARS".
**FILE**
```

o. B114: Appends B4 for 9114

```
DIVISION LABELS '04'  '94' '95'.
X AXIS LABEL IS "COST CLASS".
TITLE IS "FY 86 BUDGET VS EXPENSES"
        "COST FUNCTION 9114".
DEPENDENT LABEL TEXT IS "THOUSANDS OF DOLLARS".
**FILE**
```

p. B115: Appends B4 for 9115

```
DIVISION LABELS '02' '91' '93'.
X AXIS LABEL IS "COST CLASS".
TITLE IS "FY 86 BUDGET VS EXPENSES"
        "COST FUNCTION 9115".
DEPENDENT LABEL TEXT IS "THOUSANDS OF DOLLARS".
**FILE**
```

q. B116: Appends B4 for 9116

```
DIVISION LABELS '03' '04' '91' '93'.
X AXIS LABEL IS "COST CLASS".
TITLE IS "FY 86 BUDGET VS EXPENSES"
        "COST FUNCTION 9116".
DEPENDENT LABEL TEXT IS "THOUSANDS OF DOLLARS".
**FILE**
```

r. B117: Appends B4 for 9117

```
DIVISION LABELS '03' '12' '33' '68' '91' '93' '96' '97'.
X AXIS LABEL IS "COST CLASS".
TITLE IS "FY 86 BUDGET VS EXPENSES"
        "COST FUNCTION 9117".
```

```
DEPENDENT LABEL TEXT IS "THOUSANDS OF DOLLARS".
**FILE**
```

s. B118: Appends B4 for 9118

```
DIVISION LABELS '91' '93'.
X AXIS LABEL IS "COST CLASS".
TITLE IS "FY 86 BUDGET VS EXPENSES"
        "COST FUNCTION 9118".
DEPENDENT LABEL TEXT IS "THOUSANDS OF DOLLARS".
**FILE**
```

t. B119: Appends B4 for 9119

```
DIVISION LABELS '04'.
X AXIS LABEL IS "COST CLASS".
TITLE IS "FY 86 BUDGET VS EXPENSES"
        "COST FUNCTION 9119".
DEPENDENT LABEL TEXT IS "THOUSANDS OF DOLLARS".
**FILE**
```

u. PERBAR: Bar Chart Percent Expended

```
GENERATE A VERTICAL BAR CHART.
INDEPENDENT DIVISION-LABELS IS 'ELPSED'  'STD TIME'
       'OVER TIME' 'MATERIAL' 'OTHER' 'TOTAL'.
DEPENDENT SCALE MAXIMUM IS 100.
DEPENDENT SCALE MINIMUM IS 0.
DEPENDENT SCALE STEP-SIZE IS 20.
DEPENDENT LABEL TEXT IS "PERCENT".
TITLE TEXT IS "PERCENT EXPENDED".
BAR ROOT IS 0.
AXIS FRAME IS 1.
X AXIS ORIGIN 1.5, LENGTH 9.
Y AXIS LENGTH 1.25, ORIGIN 6.25.
SUBPLOT 1.
**FILE**
```

v. NORBAR: Bar Chart Normalized for Elapsed Time

```
GENERATE A VERTICAL BAR CHART.
INDEPENDENT DIVISION-LABELS IS 'ELAPSED'  'STD TIME'
                  'OVER TIME' 'MATERIAL' 'OTHER' 'TOTAL'.
TITLE TEXT IS "DATA NORMALIZED ON PERCENT ELAPSED TIME".
BAR ROOT IS 1.
X AXIS ORIGIN 1.5, LENGTH 9.
Y AXIS LENGTH = 1.25, ORIGIN = 4.25.
Y GRID ON.
AXIS FRAME IS 1.
SUBPLOT 2.
**FILE**
```

w. VARBAR: Bar Chart Variance in Dollars

```
GENERATE A VERTICAL BAR CHART.
INDEPENDENT DIVISION-LABELS IS 'ELAPSED'  'STD TIME'
               'OVER TIME' 'MATERIAL' 'OTHER' 'TOTAL'.
INDEPENDENT GRID IS 1.
DEPENDENT LABEL TEXT IS "DOLLARS".
TITLE TEXT IS "VARIANCE IN DOLLARS".
BAR ROOT IS 0.
X AXIS ORIGIN 1.5, LENGTH 9.
Y AXIS LENGTH = 1.25, ORIGIN = 2.25.
AXIS FRAME IS 1.
SUBPLOT 3.
**FILE**
```

x. PERVAR: Bar Chart Percent Variance

```
GENERATE A VERTICAL BAR CHART.
INDEPENDENT DIVISION-LABELS IS 'ELAPSED'  'STD TIME'
          'OVER TIME' 'MATERIAL' 'OTHER' 'TOTAL'.
INDEPENDENT GRID IS 1.
INDEPENDENT LABEL TEXT IS "PERCENT".
TITLE TEXT IS "PERCENT VARIANCE".
X AXIS ORIGIN 1.5, LENGTH 9.
Y AXIS LENGTH = 1.25, ORIGIN = .25.
```

```
BAR ROOT IS 0.
AXIS FRAME IS 1.
SUBPLOT 4.
**FILE**
```

y. B110: Data File for B1

```
INPUT DATA.
"BUD86"
1 0.77287 2 2.13234 3 4.33018 4 0.05306 5 0.27409 6 1.48263
7 0.0898 8 0.507
END OF DATA.
**FILE**
```

z. BE110: Data File for EX2

```
INPUT DATA.
"BUDGET"
 0 0 1 0.8035 12 9.64198
"EXPENSES"
0 0 1 0.92 2 1.59901 3 2.4567 4 3.3456 5 4.0002 6 4.78999 7
5.477 8 6.008 9.2 6.91127
END OF DATA.
**FILE**
```

aa. BBE110: Data File for B4

```
INPUT DATA.
"BUDGET"
1 0.77287 2 2.13234 3 4.33018 4 0.053063 5 0.274093 6 1.48263
7 0.0898 8 0.507
"BUDGET%"
 1 0.60284 2 1.66322 3 3.37754 4 0.04139 5 .21379 6 1.15645 7
.07005 8 0.39546
"EXPENSES"
 1 0.69411 2 1.3034 3 3.31009 4 0.04487 5 0.21733 6 1.10264 7
0.0665 8 0.17234
END OF DATA.
**FILE**
```

ab. PB110: Data File for Perbar

```
INPUT DATA.
"PERCENT"
1,20
2,20
3,45
4,18
5,17
6,19
END OF DATA.
**FILE**
```

ac. NB110: Data File for Norbar

```
INPUT DATA.
"NORMAL"
1,1
2,.95
3,2.2
4,0.7
5,0.55
6,0.75
END OF DATA.
**FILE**
```

ad. VB110: Data File for Varbar

```
INPUT DATA.
"VARIANCE"
1,0
2,-3000
3,1000
4,-10000
5,-70500
6,-79980
END OF DATA.
**FILE**
```

ae. PV110: Data File for Pervar

```
INPUT DATA.
"PERCENT VARIANCE"
1,0
2,-5
3,110
4,-35
5,-41
6,-30
END OF DATA.
**FILE**
```

APPENDIX E
C PROGRAMS FOR THE MICROCOMPUTER

1. CCA.C

```
#include  "colors.h"
#include          "ctype.h"
#include  "filedata.h"
#include  "intregs.h"
#include  "stdio.h"

                                /*  Global Variables */
#define BACKGRND  BLUE      /* Background color */
#define FOREGRND  YELLOW    /* Foreground color */
#define FORTY  0            /* Code for forty column mode */
#define EIGHTY  2           /* Code for eighty column mode */


extern getbud( );
extern gettotf( );
extern gettotc( );
extern gettotfc( );
extern getempjo( );
extern getemp( );
extern getfemp( );
extern getmgr( );
extern getana( );
extern getjoemp( );
extern getjoema( );
extern getfjo( );
extern getcjo( );
extern gethour( );
extern getlab( );
extern getmat( );
extern getoth( );
```

```
struct filedata filstruc;
long int _stack = 20000;

#include  "orcainp"

main( )
       /* 1  main begin  */
          char select[256],
                          *hdr;

   char
        names[21],
        cfno[5],
        clno[5],
        jono[5],
        jname[50];

   char *budget[100];
   int  i, j, pg, pflag, plines, inp[1], lines, nlines,
         inp1[1],
         inp2[1],
         inp3[1],
         inp4[1],
         inp5[1],
         inp6[1],
         inp7[1];


   short curs[2][32];      /* lda and three cursors */

/*************************************************************/
/*                  program module Cost Center Analysis      */
/*                      version 1.0                          */
/*             authors:  Richard N. Woodman                  */
/*                       Michael F Rall                      */
/*                                                           */
/*                                                           */
/*                                                           */
```

```
/*          Program last modified 20 January 1986        */
/*                                                       */
/*  This program was produced on an IBM PC using         */
/*  DOS 3.1.  Written with the C programming language,   */
/*  utilizing the GraphiC utility software.              */
/*                                                       */
/*  Main module; controls entry to the Cost Center       */
/*  Information modules or to the Command Level Entry    */
/*  mode.                                                */
/*                                                       */
/*                                                       */
/*                                                       */
/*  Input/Output Files used:  None                       */
/*                                                       */
/*  Other Modules Called: COMDLEV, CCI                   */
/*                                                       */
/*  Called by:  None                                     */
/*                                                       */
/*  Local Variables:  inp                                */
/*******************************************************/

   setscmod(EIGHTY);
   clscolor(FOREGRND,BACKGRND);
   border(BACKGRND);
    /*        */

        inp1[0] = ' 0';
        inp2[0] = ' 0';
        inp3[0] = ' 0';
        inp4[0] = ' 0';
        inp5[0] = ' 0';
        inp6[0] = ' 0';
        inp7[0] = ' 0';

                                clearthescreen();
                                clscolor(14,3);
```

```
                        riteborder();
clearkbd();
                        do
    /*   4     */

clearkbd();
inp[0] = ' 0';
                        curlocat(4,1);
                        colrcprt("COST CENTER ANALYSIS ",14,3);
                        curlocat(6,1);
                        colrcprt("1.  Cost Center Information ",14,3);
                        curlocat(8,1);
              colrcprt("2.  Cost Center Information Using  ",14,3);
                   curlocat(9,1);
           colrcprt("Oracle Command Language (SQL) For Adhoc",14,3);
                        curlocat(10,1);
                colrcprt("Queries, Inserts, Deletes and Updates",14,3);
                        curlocat(14,1);
                        colrcprt("Selection:    ",14,3);
                        curlocat(16,1);
                        colrcprt("A blank line exits to DOS   ",14,3);

                        curlocat(14,45);
                        getint(1,1,1,1,2,&inp,0,1,3);
                        if (inp[0] != ' 0')

                                curlocat(15,1);
                           colrcprt("Is this correct?          ",14,3);
                                curlocat(15,45);
                                i = ecoyesno(15,45,1);

                        if (inp[0] == ' 0')
                                i = 1;

                        while (i != 1);
```

```
            curlocat(14,45);
        /*      */
        if (inp[0] == ' 0') goto done;

        if (inp[0] == 1)


    /* Logon to ORACLE */
    if (olon(curs[0],"system/manager",-1,-1,-1,0))
                /* 2       */
        errrpt(curs[0],4);
        goto done;
              /*   2     */

/*************************************************************/
/*                  program module Cost Center Information */
/*                       version 1.0                       */
/*             authors:  Richard N. Woodman                */
/*                       Michael F Rall                    */
/*                                                         */
/*                                                         */
/*                                                         */
/*           Program last modified 20 January 1986        */
/*                                                         */
/*  This program was produced on an IBM PC  using          */
/*  DOS 3.1.  Written with the C programming language,     */
/*  utilizing the GraphiC utility software.               */
/*                                                         */
/*  Main menu driven shell for Oracle.  Allows easy access */
/*  to specified information and display of that informa-  */
/*  from Oracle.  Also sends specified data to a file for  */
/*  the Graphics utilities.                                */
/*                                                         */
/*  Output files:  GRAF, GRAF1, BUD                        */
/*                                                         */
/*  Modules called:  BUD_EXP, EMPINFO, JOINFO              */
```

```
/*                                                          */
/*  Called by: CCA                                          */
/*                                                          */
/*  Local Variables:  inp1                                  */
/************************************************************/

   for (;;)
                                    /* 3  begin main while statement  */
                                clearthescreen( );
                                clscolor(14,3);
                                riteborder( );
          clearkbd( );
                                do
              /*   4     */

          clearkbd( );
          inp1[0] = ' 0';
                                curlocat(4,1);
                                colrcprt("INFORMATION AVAILABLE ",14,3);
                                curlocat(6,1);
                                colrcprt("1.  Budget VS Expenses ",14,3);
                                curlocat(10,1);
                                colrcprt("2.  Job Order Information ",14,3);
                                curlocat(14,1);
                                colrcprt("Selection:    ",14,3);
                                curlocat(16,1);
                                colrcprt("A blank line exits     ",14,3);

                                curlocat(14,45);
                                getint(1,1,1,1,2,&inp1,0,1,3);

                                if (inp1[0] != ' 0')

                                        curlocat(15,1);
```

```
                                        colrcprt("Is this correct?
",14,3);
                                        curlocat(15,45);
                                        i = ecoyesno(15,45,1);

                                    if (inpl[0] == ' 0')
                                        i = 1;

                                    while (i != 1);

            curlocat(14,45);
        /*      */
        if (inpl[0] == ' 0') break;

/*************************************************************/
/*                  program module Budget vs Expenses        */
/*                      version 1.0                          */
/*             authors:  Richard N. Woodman                  */
/*                       Michael F Rall                      */
/*                                                           */
/*                                                           */
/*                                                           */
/*           Program last modified 20 January 1986          */
/*                                                           */
/*  This program was produced on an IBM PC using             */
/*  DOS 3.1.  Written with the C programming language,       */
/*  utilizing the GraphiC utility software.                  */
/*                                                           */
/*  Allows display and comparison of budget and actual       */
/*  expense information by various categories.  Interfaces   */
/*  specified data with graphics for further displays.       */
/*                                                           */
/*  Output files:  GRAF, GRAF1, BUD                          */
/*                                                           */
/*  Modules called:  GETBUD, INDVDISP, TOTBUDEXP             */
/*                                                           */
```

```
/*  Called by:  CCI                                          */
/*                                                           */
/*  Local Variables:  inp2                                   */
/***********************************************************/

        if (inp1[0] == 1)

           for(;;)

                                clearthescreen();
                                clscolor(14,3);
                                riteborder();
                                do

            clearkbd();
                                curlocat(4,1);
                                colrcprt("BUDGET VS EXPENSES ",14,3);
                                curlocat(6,1);
                                colrcprt("1.   Total  Budget  VS  Expenses  to  Date
",14,3);
                                curlocat(8,1);
                                colrcprt("2.  Labor or Material or Other  ",14,3);
                                curlocat(10,1);
                           colrcprt("3.  Budget by Cost Func/Cost Class",14,3);
                                curlocat(14,1);
                                colrcprt("Selection:      ",14,3);
                                curlocat(16,1);
                                colrcprt("A blank line exits  ",14,3);
                                curlocat(14,45);
           -
                                getint(1,1,1,1,2,&inp2,0,1,3);
                                if (inp2[0] != ' 0')

                                      curlocat(15,1);
                                   colrcprt("Is this correct?          ",14,3);
                                      curlocat(15,45);
```

```
                                        i = ecoyesno(15,45,1);

                                    if (inp2[0] == ' 0')
                                        i = 1;
                                    /*  END DO LOOP FOR 2ND MENU IF MENU 1 = 1   */
                                    while (i != 1);

        /*      */
        if (inp2[0] == ' 0') break;
/*************************************************************/
/*                  program module Total Budget vs Expenses*/
/*                      version 1.0                          */
/*             authors:  Richard N. Woodman                  */
/*                       Michael F Rall                      */
/*                                                           */
/*                                                           */
/*                                                           */
/*           Program last modified 20 January 1986          */
/*                                                           */
/*  This program was produced on an IBM PC    using         */
/*  DOS 3.1.  Written with the C programming language,       */
/*  utilizing the GraphiC utility software.                  */
/*                                                           */
/*  Sums labor, material and other for budget and expenses   */
/*  to date by cost function, cost class, cost function/     */
/*  cost class and cost center as requested and sends data   */
/*  to graphics routine when directed.                       */
/*                                                           */
/*  Output:  GRAF, GRAF1, BUD                                */
/*                                                           */
/*  Modules called:  GETTOTF, GETTOTC, GETTOTCF, GETSUM      */
/*                                                           */
/*  Called by: BUD_EXP                                       */
/*                                                           */
/*  Local Variables:  inp3                                   */
```

```
/*************************************************************/

      if (inp2[0] == 1)

        for(;;)

                                clearthescreen();
                                clscolor(14,3);
                                riteborder();
                                do

         clearkbd();
                                curlocat(4,1);
                                colrcprt("TOTAL BUDGET VS EXPENSES ",14,3);
                                curlocat(6,1);
                                colrcprt("1.  Cost Function ",14,3);
                                curlocat(8,1);
                                colrcprt("2.  Cost Class  ",14,3);
                                curlocat(10,1);
                                colrcprt("3.  Cost Function Cost Class ",14,3);
                                curlocat(12,1);
                                colrcprt("4.  Cost Center  ",14,3);
                                curlocat(14,1);
                                colrcprt("Selection:      ",14,3);
                                curlocat(16,1);
                                colrcprt("A blank line exits  ",14,3);
                                curlocat(14,45);
                                getint(1,1,1,1,2,&inp3,0,1,4);
                                if (inp3[0] != ' 0')

                                        curlocat(15,1);
                                    colrcprt("Is this correct?          ",14,3);
                                        curlocat(15,45);
                                        i = ecoyesno(15,45,1);
```

```
                        if (inp3[0] == ' 0')
                                i = 1;

                        while (i != 1);

        if (inp3[0] == ' 0') break;

            clscolor(FOREGRND,BACKGRND);
            curlocat(15,15);
            colrprts("Total Budget VS Expenses in Thousands of Dollars",
                                FOREGRND,BACKGRND);
            pause();

        if (inp3[0] == 1)

            hdr = " COST FUN     BUDGET       EXPENSE
In Thousands   ";
            strcpy(select, selbfun);
              gettotf(select, hdr, curs);


        if (inp3[0] == 2)

            hdr = " COST CLS     BUDGET       EXPENSE
              In Thousands    ";
            strcpy(select, selbcl);
              gettotc(select, hdr, curs);


        if (inp3[0] == 3)

            hdr = " COST FUNC    COST CLS            BUDGET       EXPENSE
                     In Thousands  ";
            strcpy(select, selbcfcl);
              gettotfc(select, hdr, curs);

        if (inp3[0] == 4)
```

```
            hdr = "COST CENTER 9110   BUDGET        EXPENSE
               DATE     In Thousands      ";
            strcpy(select, selsum);
             getsum(select, hdr, curs);



/*************************************************************/
/*                  program module INDVDISP                  */
/*                      version 1.0                          */
/*             authors:  Richard N. Woodman                  */
/*                       Michael F Rall                      */
/*                                                           */
/*                                                           */
/*                                                           */
/*           Program last modified 20 January 1986          */
/*                                                           */
/*  This program was produced on an IBM PC    using         */
/*  DOS 3.1.  Written with the C programming language,     */
/*  utilizing the GraphiC utility software.                */
/*                                                           */
/*  Display budget vs expenses to date for either labor,   */
/*  material or other, sorted by Cost Function/Cost Class. */
/*                                                           */
/*  Input/Output files:  None                              */
/*                                                           */
/*  Modules Called:  GETLAB, GETHOUR, GETMAT, GETOTH       */
/*                                                           */
/*  Local Variables:  inp4                                 */
/*************************************************************/

       if (inp2[0] == 2)
```

```
for(;;)

                        clearthescreen();
                        clscolor(14,3);
                        riteborder();
                        do

 clearkbd();
                        curlocat(4,1);
                        colrcprt(" BUDGET VS EXPENSES ",14,3);
                        curlocat(6,1);
                        colrcprt("1.  HOURS             ",14,3);
                        curlocat(8,1);
                        colrcprt("2.  LABOR             ",14,3);
                        curlocat(10,1);
                        colrcprt("3.  MATERIAL          ",14,3);
                        curlocat(12,1);
                        colrcprt("4.  OTHER          ",14,3);
                        curlocat(14,1);
                        colrcprt("Selection:      ",14,3);
                        curlocat(16,1);
                        colrcprt("A blank line exits  ",14,3);
                        curlocat(14,45);
                        getint(1,1,1,1,2,&inp4,0,1,4);
                        if (inp4[0] != ' 0')

                                    curlocat(15,1);
                                    colrcprt("Is this correct?"
                                                        ,14,3);
                                    curlocat(15,45);
                                    i = ecoyesno(15,45,1);

                        if (inp4[0] == ' 0')
                                    i = 1;
```

```
                    while (i != 1);

if (inp4[0] == ' 0') break;

     clscolor(FOREGRND,BACKGRND);
     curlocat(15,15);
     colrprts(" Budget VS Expenses in Thousands of Dollars",
                FOREGRND,BACKGRND);
     pause();



if (inp4[0] == 1)

     hdr = " HOURS: COST FUN COST CLS
                  BUDGET                    EXPENSE    ";

     strcpy(select, selhour);
      gethour(select, hdr, curs);


if (inp4[0] == 2)

     hdr = " LABOR: COST FUN COST CLS
        BUDGET              EXPENSE  In Thousands";
     strcpy(select, sellab);
      getlab(select, hdr, curs);


if (inp4[0] == 3)

     hdr = " MATERIAL: COST FUNC COST CLS
  BUDGET.             EXPENSE  In Thousands    ";
     strcpy(select, selmat);
      getmat(select, hdr, curs);


if (inp4[0] == 4)
```

```
    hdr = " OTHER: COST FUNC COST CLS
    BUDGET            EXPENSE  In Thousands";
    strcpy(select, seloth);
     getoth(select, hdr, curs);



if (inp2[0] == 3)

    clscolor(FOREGRND,BACKGRND);
    curlocat(15,20);
    colrprts("Budget By Cost Function Cost Class
                  in Thousands of Dollars",FOREGRND,BACKGRND);
    pause();

for (j = 2; j < 10; j++)

    strcpy(select, selfun);
    strcat(select,"'911");
    if (j == 2)

        hdr = "9112 COST CL        LABOR       MATERIAL
           OTHER      IN THOUSANDS     ";
        strcat(select,"2");

    if (j == 3)

        hdr = "9113 COST CL        LABOR       MATERIAL
                 OTHER      IN THOUSANDS";
        strcat(select,"3");

    if (j == 4)

        hdr = "9114 COST CL        LABOR       MATERIAL
```

```
            OTHER      IN THOUSANDS";
    strcat(select,"4");

if (j == 5)

    hdr = "9115 COST CL         LABOR       MATERIAL
        OTHER       IN THOUSANDS";
    strcat(select,"5");

if (j == 6)

    hdr = "9116 COST CL         LABOR       MATERIAL
          OTHER         IN THOUSANDS";
    strcat(select,"6");

if (j == 7)

    hdr = "9117 COST CL         LABOR       MATERIAL
          OTHER          IN THOUSANDS";
    strcat(select,"7");

if (j == 8)

    hdr = "9118 COST CL         LABOR        MATERIAL
          OTHER           IN THOUSANDS";
    strcat(select,"8");

if (j == 9)

    hdr = "9119 COST CL         LABOR        MATERIAL
          OTHER           IN THOUSANDS";
    strcat(select,"9");

strcat(select,"'");
 getbud(select, hdr, curs);
```

```
                /*  END 3RD MENU 4. IF = 4  */

          /* END SECOND MENU FOR LOOP          */
        /* END 1. BUDGET AND EXPENSE IF LOOP FROM SECOND MENU */

/*************************************************************/
/*                  program module Job Order Information     */
/*                       version 1.0                         */
/*             authors:  Richard N. Woodman                  */
/*                       Michael F Rall                      */
/*                                                           */
/*                                                           */
/*                                                           */
/*           Program last modified 20 January 1986          */
/*                                                           */
/*  This program was produced on an IBM PC     using         */
/*  DOS 3.1.  Written with the C programming language,       */
/*  utilizing the GraphiC utility software.                  */
/*                                                           */
/*  Displays the a Job Order, Job Orders                     */
/*  under a Cost Function, and Job Orders under a Cost       */
/*  Class.                                                   */
/*                                                           */
/*  Input/Output Files:  None                                */
/*                                                           */
/*  Modules Called:  GETJOEMP, GETJOEMA, GETFJO, GETCJO      */
/*                                                           */
/*  Called by:  CCI                                          */
/*                                                           */
/*  Local Variables:  inp7                                   */
/*************************************************************/

        if (inp1[0] == 2)
                                    /*   main menu choose 3  */
           for(;;)
```

```
                    clearthescreen();
                    clscolor(14,3);
                    riteborder();
inp[0] = ' 0';
                    do

clearkbd();
inp7[0] = ' 0';
                    curlocat(4,1);
                    colrcprt("JOB ORDER INFORMATION ",14,3);
                    curlocat(8,1);
                    colrcprt("1.  Input Cost Function #
                                        Find Job Orders",14,3);
                    curlocat(10,1);
                    colrcprt("2.  Input Cost Class # Find
                                            Job Orders",14,3);
                    curlocat(14,1);
                    colrcprt("Selection:      ",14,3);
                    curlocat(16,1);
                    colrcprt("A blank line exits  ",14,3);
          curlocat(14,45);
                    getint(1,1,1,1,2,&inp7,0,1,4);
                    if (inp7[0] != ' 0')

                              curlocat(15,1);
                              colrcprt("Is this correct?
                                    ",14,3);
                              curlocat(15,45);
                              i = ecoyesno(15,45,1);

                    if (inp7[0] == ' 0')
                              i = 1;
                       /*  END DO LOOP FOR MENU IF MENU 1 = 3   */
                    while (i != 1);
```

```
    curlocat(14,45);
/*      */
if (inp7[0] == ' 0') break;


        clscolor(FOREGRND,BACKGRND);
        curlocat(15,20);
        colrprts("   JOB ORDER INFORMATION   ",FOREGRND,BACKGRND);
        pause();


    if (inp7[0] == 1)


  for (;;)
                              /*  begin for statement  */
                          clearthescreen();
                          clscolor(14,3);
                          riteborder();
                          do

     clearkbd();
                          curlocat(6,1);
                          colrcprt("INPUT THE COST FUNCTION NUMBER  ",14,3);
                          curlocat(10,1);
                          colrcprt("Selection:                ",14,3);
                          curlocat(16,1);
                          colrcprt("A blank line exits   ",14,3);
                          curlocat(10,45);
                          getcstr(20,1,1,1,2,&cfno,0);
                          if (cfno[0] != ' 0')

                                  curlocat(12,1);
                                  colrcprt("Is this correct?"
                                                  ,14,3);
```

```
                                curlocat(12,45);
                                i = ecoyesno(12,45,1);

                        if (cfno[0] == ' 0')
                                i = 1;

                        while (i != 1);

        if (cfno[0] == ' 0') break;

        hdr = " Cost Function        Job Order Number
                                ";
        strcpy(select, selfjo);
        strcat(select,"'");
        strcat(select,cfno);
        strcat(select,"'");
        getfjo(select, hdr, curs);
           /*  end of or if loop  = 1 for menu 2-2  */
    /*   end of employee menu for loop  */

    if (inp7[0] == 2)

for (;;)
                                /*  begin for statement  */
                        clearthescreen();
                        clscolor(14,3);
                        riteborder();
                        do

    clearkbd();
                        curlocat(6,1);
                        colrcprt("INPUT THE COST CLASS NUMBER  ",14,3);
                        curlocat(10,1);
                        colrcprt("Selection:                  ",14,3);
```

```
                                curlocat(16,1);
                                colrcprt("A blank line exits   ",14,3);
                                curlocat(10,45);
                                getcstr(20,1,1,1,2,&clno,0);
                                if (clno[0] != ' 0')

                                        curlocat(12,1);
                                        colrcprt("Is this correct? "
                                                ,14,3);
                                        curlocat(12,45);
                                        i = ecoyesno(12,45,1);

                                if (clno[0] == ' 0')
                                        i = 1;

                                while (i != 1);

        if (clno[0] == ' 0') break;

        hdr     =     "     Cost     Class                Job     Order     Number
"};

        strcpy(select, selcjo);
        strcat(select,"'");
        strcat(select,clno);
        strcat(select,"'");
        getfjo(select, hdr, curs);
           /*  end of or if loop  = 1 for menu 2-2  */
     /*   end of employee menu for loop  */



      /*   end of for loop   */
    /*    end of if =2 employee          */
/*  END FIRST MENU FOR LOOP  */
```

```
    /*  End Main Menu If Inp == 1 */

/*************************************************************/
/*                    program module UFI                     */
/*                        version 1.0                        */
/*               authors:  Richard N. Woodman                */
/*                         Michael F Rall                    */
/*                                                           */
/*                                                           */
/*                                                           */
/*            Program last modified 20 January 1986          */
/*                                                           */
/*  This program was produced on an IBM PC     using         */
/*  DOS 3.1.  Written with the C programming language,       */
/*  utilizing the GraphiC utility software.                  */
/*                                                           */
/*  Calls User Friendly Interface to allow the user to       */
/*  make Adhoc queries, updates, and deletes.                */
/*                                                           */
/*  Input/Output Files:  None                                */
/*                                                           */
/*  Modules called: User Friendly Interface                  */
/*                                                           */
/*  Called by:  CCA                                          */
/*                                                           */
/*  Local Variables:  None                                   */
/*************************************************************/

if (inp[0] == 2)

                                    clscolor(14,3);
     i = doscmd("ufi system/manager");
      /*  End Main Menu If Inp == 2 */


   done:
```

```
/*  Close the budget cursor */
oclose(curs[1]);
/*  Free the budget array */
for(i=0; i < 50; i++) free(budget[i]);
if (pflag) lprtff();
setscmod(EIGHTY);
border(BACKGRND);
clscolor(FOREGRND,BACKGRND);


/* Log off from ORACLE */
ologof(curs[0]);
        .    /*  main end  */
curlocat(1,i);
colrprts(title,FOREGRND,BACKGRND);
return;



      riteborder()

                int x;

                nextline(2);
                colrcprt("??????????????????????????????????????????",
                                              14,3);
                printf(" n");
                for (x = 0; x <= 13; x++)

      colrcprt("?                                          ?",14,3);
                printf(" n");

                colrcprt("??????????????????????????????????????????",
                                           14,3);
```

```
        nextline(line)


                int y;
                for (y = 0; y <= line; y++)
                        printf(" n");



        clearthescreen( )

                int z;

                for (z = 4; z < 17; z++)
                clrmsg(z,20,38);



writef(line)
char line[81];
/* Writefile writes the output of Oracle to a file called graf */

     FILE *outfile, *fopen( );

          outfile = fopen("graf","a");
          fprintf(outfile,"%s n",line);
          fclose(outfile);



writef1(line)
char line[81];
/* Writefile1 writes the output of Oracle to a file called graf1 */

     FILE *outfile, *fopen( );

          outfile = fopen("graf1","a");
```

```
        fprintf(outfile,"%s n",line);
        fclose(outfile);



writefb(line)
char line[81];
/* Writefb writes the output of Oracle to a file called bud */

    FILE *outfile, *fopen( );

        outfile = fopen("bud","a");
        fprintf(outfile,"%s n",line);
        fclose(outfile);
```

2. PROJA.C

```
#include  "colors.h"
#include        "ctype.h"
#include  "filedata.h"
#include  "intregs.h"
#include  "stdio.h"

                            /*  Global Variables */
#define BACKGRND  BLUE     /* Background color */
#define FOREGRND  YELLOW   /* Foreground color */
#define FORTY  0           /* Code for forty column mode */
#define EIGHTY  2          /* Code for eighty column mode */



#include  "orcainp"


/*************************************************************/
```

```
/*                  program module getbud                    */
/*                      version 1.0                          */
/*              authors:  Richard N. Woodman                 */
/*                        Michael F Rall                     */
/*                                                           */
/*                                                           */
/*                                                           */
/*           Program last modified 11 December 1986          */
/*                                                           */
/*  Purpose:  Displays budget summary for current fiscal     */
/*  year, by cost function / cost class.                     */
/*                                                           */
/*  Other modules called:  SELFUN                            */
/*                                                           */
/*  Called by: BUD_EXP                                       */
/*                                                           */
/*                                                           */
/*  Files used:  NONE                                        */
/*                                                           */
/*                                                           */
/*  Local variables: clno,labor,material,other,              */
/*      line-80-, *budget-100-, *calloc( );                  */
/*      i, j, pg, pflag, plines, lines, nlines;              */
/*                                                           */
/*************************************************************/


getbud(select, hdr, curs)
   char *select,
          *hdr;
   short curs[ ][32];

     /*  BEGIN GETBUDGET ROUTINE   */
   char
       clno[3],
       labor[15],
```

```
    material[15],
    other[15];

char line[80], *budget[100], *calloc();
int  i, j, pg, pflag, plines, lines, nlines;


   nlines = 0;

setscmod(EIGHTY);
clscolor(FOREGRND,BACKGRND);
border(BACKGRND);
curlocat(12,23);
for(i=0; i < 50; i++) budget[i] = calloc(1,1);
colrprts("Do you want printed output (Y/N)?",FOREGRND,BACKGRND);
if (getyesno(1))

    /*  Initialize the print variables */
    pflag = 1;
    for (j=1; j <= 6; j++) lprtlf();
    for (j=1; j <= 1; j++) lprtchar(0, ' ');
    lprtstr(hdr);
    lprtcr();
    lprtlf();
    lprtlf();


else pflag = 0;
plines = 0;

  clscolor(FOREGRND,BACKGRND);
/* Process the ORACLE request */
/*  Open a cursor for the budget */
if (oopen(curs[1],curs[0],-1,-1,-1,-1,-1))

    errrpt(curs[0],4);
    goto close;
```

```
/*  Retrieve the first record */
      /*SELECT SUM(LABOR)+SUM(MATERIAL)+SUM(OTHER)
          FROM BUDGET WHERE COST_FUN_NO =  */
if (osql3(curs[1], select, -1) ||
odefin(curs[1], 1, &clno,  sizeof clno,
   5, -1,-1,-1,-1,-1,-1,-1) ||
odefin(curs[1], 2, &labor,   sizeof labor,
   5, -1,-1,-1,-1,-1,-1,-1) ||
odefin(curs[1], 3, &material,   sizeof material,
   5, -1,-1,-1,-1,-1,-1,-1) ||
odefin(curs[1], 4, &other,  sizeof other,  5, -1,-1,-1,-1,-1,-1,-1) ||
                oexec(curs[1]) ||
     ofetch(curs[1]))

     if(curs[1][0]==4)

          curlocat(12,30);
          colrprts("No Records Selected",FOREGRND,BACKGRND);
          goto close;

     else

          errrpt(curs[0],4);
          goto close;



/*  Retrieve the remaining records */
lines = 2;
head(hdr);
while (curs[1][0] != 4)

     nlines = 0;
     strcpy(line, clno);
```

```
strcat(line, " ");
strcat(line, labor);
strcat(line, " ");
strcat(line, material);
strcat(line, " ");
strcat(line, other);

if (strcmp(line, " ") != 0)

    free(budget[nlines]);
    j = strlen(line);
    budget[nlines] = calloc(j+1,1);
    strcpy(budget[nlines], line);
    nlines++;




/* Check for a very large entry */
if (nlines > 21)

    clscolor(FOREGRND,BACKGRND);
    curlocat(12,24);
    colrprts("**********Entry exceeds 20 lines",
        FOREGRND,BACKGRND);
    curlocat(24,21);
    colrprts("Press any key to continue or Q to quit",
        FOREGRND,BACKGRND);
    getkey(&j);
    if ((j == 'q') || (j == 'Q')) goto done;
    clscolor(FOREGRND,BACKGRND);
    lines = 2;
    for (j=0; j < nlines; j++) printf("         %s n",budget[j]);
```

```
/* Check for a full screen */
else if (lines + nlines > 23)

     curlocat(24,21);
     colrprts("Press any key to continue or Q to quit",
          FOREGRND,BACKGRND);
     getkey(&j);
     lines = 2;
     if ((j == 'q') || (j == 'Q')) goto done;
     clscolor(FOREGRND,BACKGRND);
     head(hdr);


/* Check for a full page */
if (pflag && ((plines + nlines) > 51))

     plines = 0;
     i = (80 - strlen(hdr))/2;
     lprtff();
     for (j=1; j <= 6; j++) lprtlf();
     for (j=1; j <= i; j++) lprtchar(0, ' ');
     lprtstr(hdr);
     lprtcr();
     lprtlf();
     lprtlf();


/* Print the lines */
for(j=0; j < nlines; j++)

     curlocat(lines,10);
     colrprts(budget[j],FOREGRND,BACKGRND);
     if (pflag)
```

```
                for (j=1; j <= 9; j++) lprtchar(0, ' ');
                lprtstr(budget[j]);
                lprtcr();
                lprtlf();
                plines++;

            lines++;

        lines++;
        if (pflag)

            lprtcr();
            lprtlf();
            plines++;

ofetch(curs[1]);

  close:
  curlocat(24,20);
  colrprts("        Press a key to continue          ",
                    FOREGRND,BACKGRND);
  pause();

  done:

  /*  Close the budget cursor */
  oclose(curs[1]); /*  Free the budget array */
  for(i=0; i < 50; i++) free(budget[i]);
  if (pflag) lprtff();
  setscmod(EIGHTY);
  border(BACKGRND);
  clscolor(FOREGRND,BACKGRND);
```

```
/*************************************************************/
/*                  program module gettotf                   */
/*                      version 1.0                          */
/*             authors:  Richard N. Woodman                  */
/*                       Michael F Rall                      */
/*                                                           */
/*                                                           */
/*                                                           */
/*           Program last modified 11 December 1986          */
/*                                                           */
/*  Purpose:  Displays budget vs expense to date for         */
/*  current fiscal year, by cost function.                   */
/*                                                           */
/*  Other modules called:  SELBFUN,SELEFUN,writef            */
/*                                                           */
/*  Called by: TOBUDEXP                                      */
/*                                                           */
/*                                                           */
/*  Files used:  NONE                                        */
/*  Files created:  GRAF                                     */
/*                                                           */
/*                                                           */
/*  Local variables: cfno,bud,exp                            */
/*      line-80-, *budget-100-, *calloc( );                  */
/*      i, j, pg, pflag, plines, lines, nlines;              */
/*                                                           */
/*************************************************************/


gettotf(select, hdr, curs)
   char *select,
          *hdr;
```

```
short curs[][32];


char
     cfno[5],
     bud[15],
     exp[15];

char line[80], *budget[100], *calloc();
int  i, j, pg, pflag, gflag, plines, lines, nlines;


   nlines = 0;

setscmod(EIGHTY);
clscolor(FOREGRND,BACKGRND);
border(BACKGRND);
curlocat(12,23);
for(i=0; i < 50; i++) budget[i] = calloc(1,1);
colrprts("Do you want printed output (Y/N)?",FOREGRND,BACKGRND);
if (getyesno(1))

     /*  Initialize the print variables */
     pflag = 1;
     for (j=1; j <= 6; j++) lprtlf();
     for (j=1; j <= 1; j++) lprtchar(0, ' ');
     lprtstr(hdr);
     lprtcr();
     lprtlf();
     lprtlf();


else pflag = 0;
plines = 0;

curlocat(14,23);
colrprts("Graph Output (Y/N)?",FOREGRND,BACKGRND);
```

```
/*  Initialize the graph variable */
if (getyesno(1)) gflag = 1;
else gflag = 0;

  clscolor(FOREGRND,BACKGRND);
/* Process the ORACLE request */
/*  Open a cursor for the budget */
if (oopen(curs[1],curs[0],-1,-1,-1,-1,-1))

     errrpt(curs[0],4);
     goto close;


/*  Retrieve the first record */
   /* SELECT COST_FUN_NO, SUM(LABOR)+SUM(MATERIAL)+SUM(OTHER),
   FROM BUDGET GROUP BY COST_FUN_NO */

if (osql3(curs[1], select, -1) ||
odefin(curs[1], 1, &cfno,  sizeof cfno,
   5, -1,-1,-1,-1,-1,-1,-1) ||
odefin(curs[1], 2, C.3,   sizeof bud,
   5, -1,-1,-1,-1,-1,-1,-1) ||
                oexec(curs[1]) ||
     ofetch(curs[1]))

     if(curs[1][0]==4)

          curlocat(12,30);
          colrprts("No Records Selected",FOREGRND,BACKGRND);
          goto close;

     else

          errrpt(curs[0],4);
          goto close;
```

```
/* Open a cursor for the expense */
/* SELECT COST_FUN_NO,SUM(LABOR)+SUM(MATERIAL)+SUM(OTHER)
   FROM EXPENSE WHERE DT = (SELECT MAX(DT) FROM EXPENSE
   AND COST_FUN_NO = :COST_FUN_NO  GROUP BY COST_FUN_NO */
    if (oopen(curs[2],curs[0],-1,-1,-1,-1,-1) ||
    osql3(curs[2], selefun, -1) ||
    odefin(curs[2], 1, &cfno,    sizeof cfno,
       5, -1,-1,-1,-1,-1,-1,-1) ||
    odefin(curs[2], 2, &exp,     sizeof exp,
       5, -1,-1,-1,-1,-1,-1,-1) ||
    obndrv(curs[2],":CFNO",-1,&cfno,-1,1,-1,-1,-1,-1))

    errrpt(curs[0],4);
    goto close;


/*  Retrieve the remaining records */
lines = 2;
head(hdr);
while (curs[1][0] != 4)

    nlines = 0;
    strcpy(line, cfno);
    strcat(line, bud);


    /*  Retrieve the first address record */
    if (oexec(curs[2]) ||
        ofetch(curs[2]))

        if(curs[2][0]==4) ;
        else

            errrpt(curs[0],4);
            goto close;
```

```
while (curs[2][0] != 4)

    strcat(line, "  ");
    strcat(line, exp);
    if (strcmp(line, " ") != 0)

        free(budget[nlines]);
        j = strlen(line);
        budget[nlines] = calloc(j+1,1);
        strcpy(budget[nlines], line);
        if (gflag == 1) writef(line);
        nlines++;


    ofetch(curs[2]);

/* Check for a very large entry */
if (nlines > 21)

    clscolor(FOREGRND,BACKGRND);
    curlocat(12,24);
    colrprts("**********Entry exceeds 20 lines",
        FOREGRND,BACKGRND);
    curlocat(24,21);
    colrprts("Press any key to continue or Q to quit",
        FOREGRND,BACKGRND);
    getkey(&j);
    if ((j == 'q') || (j == 'Q')) goto done;
    clscolor(FOREGRND,BACKGRND);
    lines = 2;
    for (j=0; j < nlines; j++) printf("          %s n",budget[j]);
```

```
/* Check for a full screen */
else if (lines + nlines > 23)

     curlocat(24,21);
     colrprts("Press any key to continue or Q to quit",
          FOREGRND,BACKGRND);
     getkey(&j);
     lines = 2;
     if ((j == 'q') || (j == 'Q')) goto done;
     clscolor(FOREGRND,BACKGRND);
     head(hdr);


/* Check for a full page */
if (pflag && ((plines + nlines) > 51))

     plines = 0;
     i = (80 - strlen(hdr))/2;
     lprtff();
     for (j=1; j <= 6; j++) lprtlf();
     for (j=1; j <= i; j++) lprtchar(0, ' ');
     lprtstr(hdr);
     lprtcr();
     lprtlf();
     lprtlf();


/* Print the lines */
for(j=0; j < nlines; j++)

     curlocat(lines,10);
     colrprts(budget[j],FOREGRND,BACKGRND);
     if (pflag)

          for (j=1; j <= 9; j++) lprtchar(0, ' ');
```

```
                lprtstr(budget[j]);
                lprtcr();
                lprtlf();
                plines++;

            lines++;

        lines++;
        if (pflag)

            lprtcr();
            lprtlf();
            plines++;



ofetch(curs[1]);

        if (gflag == 1)

            i = execute2("d:  c  graf  tripbar.exe");


    close:
    curlocat(24,20);
    colrprts("        Press a key to continue          ",
                     FOREGRND,BACKGRND);
    pause();

    done:

    /*  Close the budget cursor */
    oclose(curs[1]);
    /*  Free the budget array */
    for(i=0; i < 50; i++) free(budget[i]);
```

```
   if (pflag) lprtff( );
   setscmod(EIGHTY);
   border(BACKGRND);
   clscolor(FOREGRND,BACKGRND);



/*************************************************************/
/*                  program module gettotc                   */
/*                      version 1.0                          */
/*             authors:  Richard N. Woodman                  */
/*                       Michael F Rall                      */
/*                                                           */
/*                                                           */
/*                                                           */
/*          Program last modified 11 December 1986           */
/*                                                           */
/*  Purpose:  Displays budget vs expense to date for         */
/*  current fiscal year, by cost class.                      */
/*                                                           */
/*  Other modules called:  SELBCL,SELECL,writef              */
/*                                                           */
/*  Called by: TOBUDEXP                                     */
/*                                                           */
/*                                                           */
/*  Files used:  NONE                                        */
/*  Files created:  GRAF                                     */
/*                                                           */
/*                                                           */
/*  Local variables: clno,bud,exp                            */
/*      line-80-, *budget-100-, *calloc( );                  */
/*      i, j, pg, pflag, plines, lines, nlines;              */
/*                                                           */
/*************************************************************/
```

```
gettotc(select, hdr, curs)
   char *select,
          *hdr;
   short curs[][32];


   char
       clno[3],
       bud[15],
       exp[15];

   char line[80], *budget[100], *calloc();
   int  i, j, pg, pflag, gflag, plines, lines, nlines;


      nlines = 0;

   setscmod(EIGHTY);
   clscolor(FOREGRND,BACKGRND);
   border(BACKGRND);
   curlocat(12,23);
   for(i=0; i < 50; i++) budget[i] = calloc(1,1);
   colrprts("Do you want printed output (Y/N)?",FOREGRND,BACKGRND);
   if (getyesno(1))

       /*  Initialize the print variables */
       pflag = 1;
       for (j=1; j <= 6; j++) lprtlf();
       for (j=1; j <= 1; j++) lprtchar(0, ' ');
       lprtstr(hdr);
       lprtcr();
       lprtlf();
       lprtlf();


   else pflag = 0;
   plines = 0;
```

```
/*  Initialize the graph variable */
curlocat(14,23);
colrprts("Graph Output (Y/N)?",FOREGRND,BACKGRND);
if (getyesno(1)) gflag = 1;
else gflag = 0;

 clscolor(FOREGRND,BACKGRND);
/* Process the ORACLE request */
/*  Open a cursor for the budget */
if (oopen(curs[1],curs[0],-1,-1,-1,-1,-1))

     errrpt(curs[0],4);
     goto close;


/*  Retrieve the first record */
   /* SELECT COST_CL_NO, SUM(LABOR)+SUM(MATERIAL)+SUM(OTHER),
   FROM BUDGET GROUP BY COST_CL_NO */

if (osql3(curs[1], select, -1) ||
odefin(curs-1-, 1, &clno,  sizeof clno,  5, -1,-1,-1,-1,-1,-1,-1) ||
odefin(curs[1], 2, C.3,   sizeof bud,   5, -1,-1,-1,-1,-1,-1,-1) ||
                oexec(curs[1]) ||
     ofetch(curs[1]))

     if(curs[1][0]==4)

          curlocat(12,30);
          colrprts("No Records Selected",FOREGRND,BACKGRND);
          goto close;

     else

          errrpt(curs[0],4);
          goto close;
```

```
/* Open a cursor for the expense */
/* SELECT COST_CL_NO,SUM(LABOR)+SUM(MATERIAL)+SUM(OTHER)
   FROM EXPENSE WHERE DT = (SELECT MAX(DT) FROM EXPENSE)
   AND COST_CL_NO = :COST_CL_NO GROUP BY COST_CL_NO */
if (oopen(curs[2],curs[0],-1,-1,-1,-1,-1) ||
     osql3(curs[2], selec1, -1) ||
     odefin(curs[2], 1, &clno,    sizeof clno,
        5, -1,-1,-1,-1,-1,-1,-1) ||
     odefin(curs[2], 2, &exp,     sizeof exp,
        5, -1,-1,-1,-1,-1,-1,-1) ||
     obndrv(curs[2],":CLNO",-1,&clno,-1,1,-1,-1,-1,-1))

     errrpt(curs[0],4);
     goto close;


/*  Retrieve the remaining records */
lines = 2;
head(hdr);
while (curs[1][0] != 4)

     nlines = 0;
     strcpy(line, clno);
     strcat(line, " ");
     strcat(line, bud);
     strcat(line, " ");


     /*  Retrieve the first expense record */
     if (oexec(curs[2]) ||
          ofetch(curs[2]))

          if(curs[2][0]==4) ;
          else
```

```
        errrpt(curs[0],4);
        goto close;


while (curs[2][0] != 4)

    strcat(line, exp);
    if (strcmp(line, " ") != 0)

        free(budget[nlines]);
        j = strlen(line);
        budget[nlines] = calloc(j+1,1);
        strcpy(budget[nlines], line);
        if (gflag == 1) writef(line);
        nlines++;


    ofetch(curs[2]);


/* Check for a very large entry */
if (nlines > 21)

    clscolor(FOREGRND,BACKGRND);
    curlocat(12,24);
    colrprts("**********Entry exceeds 20 lines",
        FOREGRND,BACKGRND);
    curlocat(24,21);
    colrprts("Press any key to continue or Q to quit",
        FOREGRND,BACKGRND);
    getkey(&j);
    if ((j == 'q') || (j == 'Q')) goto done;
    clscolor(FOREGRND,BACKGRND);
    lines = 2;
    for (j=0; j < nlines; j++) printf("          %s n",budget[j]);
```

```
/* Check for a full screen */
else if (lines + nlines > 23)

     curlocat(24,21);
     colrprts("Press any key to continue or Q to quit",
          FOREGRND,BACKGRND);
     getkey(&j);
     lines = 2;
     if ((j == 'q') || (j == 'Q')) goto done;
     clscolor(FOREGRND,BACKGRND);
     head(hdr);


/* Check for a full page */
if (pflag && ((plines + nlines) > 51))

     plines = 0;
     i = (80 - strlen(hdr))/2;
     lprtff();
     for (j=1; j <= 6; j++) lprtlf();
     for (j=1; j <= i; j++) lprtchar(0, ' ');
     lprtstr(hdr);
     lprtcr();
     lprtlf();
     lprtlf();


/* Print the lines */
for(j=0; j < nlines; j++)

     curlocat(lines,10);
     colrprts(budget[j],FOREGRND,BACKGRND);
     if (pflag)
```

```
                for (j=1; j <= 9; j++) lprtchar(0, ' ');
                lprtstr(budget[j]);
                lprtcr( );
                lprtlf( );
                plines++;

            lines++;


        lines++;
        if (pflag)

            lprtcr( );
            lprtlf( );
            plines++;


        if (gflag == 1)

            i = execute2("d:lyons6.exe","lyons6");


ofetch(curs[1]);


  close:
  curlocat(24,20);
  colrprts("        Press a key to continue          ",
                   FOREGRND,BACKGRND);
  pause( );

  done:

  /*  Close the budget cursor */
  oclose(curs[1]);
  /*  Free the budget array */
  for(i=0; i < 50; i++) free(budget[i]);
  if (pflag) lprtff( );
```

```
   setscmod(EIGHTY);
   border(BACKGRND);
   clscolor(FOREGRND,BACKGRND);


/*************************************************************/
/*                  program module gettotfc                  */
/*                      version 1.0                          */
/*             authors:  Richard N. Woodman                  */
/*                       Michael F Rall                      */
/*                                                           */
/*                                                           */
/*                                                           */
/*       Program last modified 11 December 1986              */
/*                                                           */
/*  Purpose:  Displays budget vs expense to date for         */
/*  current fiscal year, by cost function/cost class.        */
/*                                                           */
/*  Other modules called:  SELBCFCL,SELECFCL                 */
/*                                                           */
/*  Called by: TOBUDEXP                                     */
/*                                                           */
/*                                                           */
/*  Files used:  NONE                                        */
/*                                                           */
/*                                                           */
/*  Local variables: cfno,clno,bud,exp                       */
/*      line-80-, *budget-100-, *calloc();                   */
/*      i, j, pg, pflag, plines, lines, nlines;              */
/*                                                           */
/*************************************************************/


gettotfc(select, hdr, curs)
   char *select,
```

```
        *hdr;
short curs[][32];


char
     cfno[5],
     clno[3],
     bud[15],
     exp[15];

char line[80], *budget[100], *calloc();
int  i, j, pg, pflag, plines, lines, nlines;


   nlines = 0;

setscmod(EIGHTY);
clscolor(FOREGRND,BACKGRND);
border(BACKGRND);
curlocat(12,23);
for(i=0; i < 50; i++) budget[i] = calloc(1,1);
colrprts("Do you want printed output (Y/N)?",FOREGRND,BACKGRND);
if (getyesno(1))

     /*  Initialize the print variables */
     pflag = 1;
     for (j=1; j <= 6; j++) lprtlf();
     for (j=1; j <= 1; j++) lprtchar(0, ' ');
     lprtstr(hdr);
     lprtcr();
     lprtlf();
     lprtlf();

else pflag = 0;
plines = 0;

  clscolor(FOREGRND,BACKGRND);
```

```
/* Process the ORACLE request */
/*  Open a cursor for the budget */
if (oopen(curs[1],curs[0],-1,-1,-1,-1,-1))

    errrpt(curs[0],4);
    goto close;


/*  Retrieve the first record */
   /* SELECT COST_FUN_NO, COST_CL_NO, LABOR+MATERIAL+OTHER
   FROM BUDGET WHERE LABOR != 0 OR MATERIAL != 0 OR OTHER != 0 */

if (osql3(curs[1], select, -1) ||
odefin(curs-1-, 1, &cfno,  sizeof cfno, 5, -1,-1,-1,-1,-1,-1,-1) ||
odefin(curs-1-, 2, &clno,  sizeof clno, 5, -1,-1,-1,-1,-1,-1,-1) ||
odefin(curs-1-, 3, C.3,   sizeof bud, 5, -1,-1,-1,-1,-1,-1,-1) ||
                oexec(curs[1]) ||
    ofetch(curs[1]))

    if(curs[1][0]==4)

        curlocat(12,30);
        colrprts("No Records Selected",FOREGRND,BACKGRND);
        goto close;

    else

        errrpt(curs[0],4);
        goto close;


/* Open a cursor for the expense */
/* SELECT COST_FUN_NO, COST_CL_NO, LABOR+MATERIAL+OTHER FROM EXPENSE
    WHERE DT = (SELECT MAX(DT) FROM EXPENSE) AND
                    (LABOR != 0 OR MATERIAL != 0 OR OTHER != 0)
    AND COST_FUN_NO = :CFNO AND COST_CL_NO = :CLNO */
```

```
if (oopen(curs[2],curs[0],-1,-1,-1,-1,-1) ||
    osql3(curs[2], selecfcl, -1) ||
    odefin(curs[2], 1, &cfno,    sizeof cfno,
       5, -1,-1,-1,-1,-1,-1,-1) ||
    odefin(curs[2], 2, &clno,    sizeof clno,
       5, -1,-1,-1,-1,-1,-1,-1) ||
    odefin(curs[2], 3, &exp,     sizeof exp,
       5, -1,-1,-1,-1,-1,-1,-1) ||
    obndrv(curs[2],":CFNO",-1,&cfno,-1,1,-1,-1,-1,-1) ||
    obndrv(curs[2],":CLNO",-1,&clno,-1,1,-1,-1,-1,-1))

    errrpt(curs[0],4);
    goto close;

/*  Retrieve the remaining records */
lines = 2;
head(hdr);
while (curs[1][0] != 4)

    nlines = 0;
    strcpy(line, cfno);
    strcat(line, "      ");
    strcat(line, clno);
    strcat(line, "    ");
    strcat(line, bud);
    strcat(line, " ");

    /*  Retrieve the first expense record */
    if (oexec(curs[2]) ||
        ofetch(curs[2]))

        if(curs[2][0]==4) ;
        else
```

```
        errrpt(curs[0],4);
        goto close;


while (curs[2][0] != 4)

    strcat(line, exp);
    if (strcmp(line, " ") != 0)

        free(budget[nlines]);
        j = strlen(line);
        budget[nlines] = calloc(j+1,1);
        strcpy(budget[nlines], line);
        nlines++;


    ofetch(curs[2]);


/* Check for a very large entry */
if (nlines > 21)

    clscolor(FOREGRND,BACKGRND);
    curlocat(12,24);
    colrprts("**********Entry exceeds 20 lines",
        FOREGRND,BACKGRND);
    curlocat(24,21);
    colrprts("Press any key to continue or Q to quit",
        FOREGRND,BACKGRND);
    getkey(&j);
    if ((j == 'q') || (j == 'Q')) goto done;
    clscolor(FOREGRND,BACKGRND);
    lines = 2;
    for (j=0; j < nlines; j++) printf("         %s n",budget[j]);
```

```
/* Check for a full screen */
else if (lines + nlines > 23)

     curlocat(24,21);
     colrprts("Press any key to continue or Q to quit",
          FOREGRND,BACKGRND);
     getkey(&j);
     lines = 2;
     if ((j == 'q') || (j == 'Q')) goto done;
     clscolor(FOREGRND,BACKGRND);
     head(hdr);


/* Check for a full page */
if (pflag && ((plines + nlines) > 51))

     plines = 0;
     i = (80 - strlen(hdr))/2;
     lprtff();
     for (j=1; j <= 6; j++) lprtlf();
     for (j=1; j <= i; j++) lprtchar(0, ' ');
     lprtstr(hdr);
     lprtcr();
     lprtlf();
     lprtlf();


/* Print the lines */
for(j=0; j < nlines; j++)

     curlocat(lines,10);
     colrprts(budget[j],FOREGRND,BACKGRND);
     if (pflag)

          for (j=1; j <= 9; j++) lprtchar(0, ' ');
```

```
                lprtstr(budget[j]);
                lprtcr();
                lprtlf();
                plines++;

            lines++;

        lines++;
        if (pflag)

            lprtcr();
            lprtlf();
            plines++;

ofetch(curs[1]);


 close:
 curlocat(24,20);
 colrprts("        Press a key to continue          ",
            FOREGRND,BACKGRND);
 pause();

 done:

 /*  Close the budget cursor */
 oclose(curs[1]);
 /*  Free the budget array */
 for(i=0; i < 50; i++) free(budget[i]);
 if (pflag) lprtff();
 setscmod(EIGHTY);
 border(BACKGRND);
 clscolor(FOREGRND,BACKGRND);


/************************************************************/
```

```
/*                  program module getfjo                    */
/*                      version 1.0                          */
/*              authors:  Richard N. Woodman                 */
/*                        Michael F Rall                     */
/*                                                           */
/*                                                           */
/*                                                           */
/*           Program last modified 20 January 1986          */
/*                                                           */
/*  Purpose:  Given cost function number, finds all          */
/*  job order numbers under if.                              */
/*                                                           */
/*  Other modules called:  SELFJO                            */
/*                                                           */
/*  Called by: JOINFO                                        */
/*                                                           */
/*                                                           */
/*  Files used:  NONE                                        */
/*                                                           */
/*                                                           */
/*  Local variables:  cfno,clno,jono                         */
/*      line-80-, *jobord-100-, *calloc( );                  */
/*      i, j, pg, pflag, plines, lines, nlines;              */
/*                                                           */
/*************************************************************/


getfjo(select, hdr, curs)
   char *select,
          *hdr;
   short curs[][32];

     /*  BEGIN GETBUDGET ROUTINE   */
   char
       cfno[5],
       clno[5],
```

```
    jono[5];

char line[80], *jobord[100], *calloc();
int  i, j, pg, pflag, plines, lines, nlines;


   nlines = 0;

setscmod(EIGHTY);
clscolor(FOREGRND,BACKGRND);
border(BACKGRND);
curlocat(12,23);
for(i=0; i < 50; i++) jobord[i] = calloc(1,1);
colrprts("Do you want printed output (Y/N)?",FOREGRND,BACKGRND);
if (getyesno(1))

    /*  Initialize the print variables */
    pflag = 1;
    for (j=1; j <= 6; j++) lprtlf();
    for (j=1; j <= 1; j++) lprtchar(0, ' ');
    lprtstr(hdr);
    lprtcr();
    lprtlf();
    lprtlf();

else pflag = 0;
plines = 0;

  clscolor(FOREGRND,BACKGRND);
/* Process the ORACLE request */
/*  Open a cursor for the jobord */
if (oopen(curs[1],curs[0],-1,-1,-1,-1,-1))

    errrpt(curs[0],4);
    goto close;
```

```
/*  Retrieve the first record */
      /*SELECT COST_FUN_NO, COST_CL_NO, JOB_ORD_NO
           FROM JOB_ORD WHERE COST_FUN_NO =  */
if (osql3(curs[1], select, -1) ||
odefin(curs-1-, 1, &cfno,  sizeof cfno, 5, -1,-1,-1,-1,-1,-1,-1) ||
odefin(curs-1-, 2, &clno,   sizeof clno, 5, -1,-1,-1,-1,-1,-1,-1) ||
odefin(curs-1-, 3, &jono,   sizeof jono, 5, -1,-1,-1,-1,-1,-1,-1) ||
                oexec(curs[1]) ||
    ofetch(curs[1]))

    if(curs[1][0]==4)

        curlocat(12,30);
        colrprts("No Records Selected",FOREGRND,BACKGRND);
        goto close;

    else

        errrpt(curs[0],4);
        goto close;



/*  Retrieve the remaining records */
lines = 2;
head(hdr);
while (curs[1][0] != 4)

    nlines = 0;
    strcpy(line, cfno);
    strcat(line, "             ");
    strcat(line, cfno);
    strcat(line, " ");
    strcat(line, clno);
    strcat(line, " ");
```

```
strcat(line, jono);

if (strcmp(line, " ") != 0)

     free(jobord[nlines]);
     j = strlen(line);
     jobord[nlines] = calloc(j+1,1);
     strcpy(jobord[nlines], line);
     nlines++;


/* Check for a very large entry */
if (nlines > 21)

     clscolor(FOREGRND,BACKGRND);
     curlocat(12,24);
     colrprts("**********Entry exceeds 20 lines",
          FOREGRND,BACKGRND);
     curlocat(24,21);
     colrprts("Press any key to continue or Q to quit",
          FOREGRND,BACKGRND);
     getKey(&j);
     if ((j == 'q') || (j == 'Q')) goto done;
     clscolor(FOREGRND,BACKGRND);
     lines = 2;
     for (j=0; j < nlines; j++) printf("          %s n",jobord[j]);



/* Check for a full screen */
else if (lines + nlines > 23)

     curlocat(24,21);
     colrprts("Press any key to continue or Q to quit",
          FOREGRND,BACKGRND);
     getKey(&j);
```

```
        lines = 2;
        if ((j == 'q') || (j == 'Q')) goto done;
        clscolor(FOREGRND,BACKGRND);
        head(hdr);


    /* Check for a full page */
    if (pflag && ((plines + nlines) > 51))

        plines = 0;
        i = (80 - strlen(hdr))/2;
        lprtff();
        for (j=1; j <= 6; j++) lprtlf();
        for (j=1; j <= i; j++) lprtchar(0, ' ');
        lprtstr(hdr);
        lprtcr();
        lprtlf();
        lprtlf();


    /* Print the lines */
    for(j=0; j < nlines; j++)

        curlocat(lines,10);
        colrprts(jobord[j],FOREGRND,BACKGRND);
        if (pflag)

            for (j=1; j <= 9; j++) lprtchar(0, ' ');
            lprtstr(jobord[j]);
            lprtcr();
            lprtlf();
            plines++;

        lines++;
```

```
        lines++;
        if (pflag)

                lprtcr( );
                lprtlf( );
                plines++;

ofetch(curs[1]);


  close:
  curlocat(24,20);
  colrprts("        Press a key to continue          ",
                FOREGRND,BACKGRND);
  pause( );

  done:

  /*  Close the jobord cursor */
  oclose(curs[1]); /*  Free the jobord array */
  for(i=0; i < 50; i++) free(jobord[i]);
  if (pflag) lprtff( );
  setscmod(EIGHTY);
  border(BACKGRND);
  clscolor(FOREGRND,BACKGRND);




/*************************************************************/
/*                  program module getcjo                    */
/*                      version 1.0                          */
/*             authors:  Richard N. Woodman                  */
/*                       Michael F Rall                      */
/*                                                           */
/*                                                           */
```

```
/*                                                          */
/*          Program last modified 20 January 1986           */
/*                                                          */
/*  Purpose:  Displays job order numbers when given a       */
/*  cost class number.                                      */
/*                                                          */
/*  Other modules called:  SELCJO                           */
/*                                                          */
/*  Called by: JOINFO                                       */
/*                                                          */
/*                                                          */
/*  Files used:  NONE                                       */
/*                                                          */
/*                                                          */
/*  Local variables: cfno,clno,jono                         */
/*      line-80-, *jobord-100-, *calloc( );                 */
/*      i, j, pg, pflag, plines, lines, nlines;             */
/*                                                          */
/************************************************************/


getcjo(select, hdr, curs)
   char *select,
          *hdr;
   short curs[ ][32];

     /*  BEGIN GETBUDGET ROUTINE   */
   char
       cfno[5],
       clno[5],
       jono[5];

   char line[80], *jobord[100], *calloc( );
   int  i, j, pg, pflag, plines, lines, nlines;


      nlines = 0;
```

```
setscmod(EIGHTY);
clscolor(FOREGRND,BACKGRND);
border(BACKGRND);
curlocat(12,23);
for(i=0; i < 50; i++) jobord[i] = calloc(1,1);
colrprts("Do you want printed output (Y/N)?",FOREGRND,BACKGRND);
if (getyesno(1))

    /*  Initialize the print variables */
    pflag = 1;
    for (j=1; j <= 6; j++) lprtlf();
    for (j=1; j <= 1; j++) lprtchar(0, ' ');
    lprtstr(hdr);
    lprtcr();
    lprtlf();
    lprtlf();

else pflag = 0;
plines = 0;

  clscolor(FOREGRND,BACKGRND);
/* Process the ORACLE request */
/*  Open a cursor for the jobord */
if (oopen(curs[1],curs[0],-1,-1,-1,-1,-1))

    errrpt(curs[0],4);
    goto close;


/*  Retrieve the first record */
      /*SELECT COST_FUN_NO, COST_CL_NO, JOB_ORD_NO
          FROM JOB_ORD WHERE COST_CL_NO =  */
if (osql3(curs[1], select, -1) ||
odefin(curs-1-, 1, &cfno,  sizeof cfno, 5, -1,-1,-1,-1,-1,-1,-1) ||
odefin(curs-1-, 2, &clno,   sizeof clno, 5, -1,-1,-1,-1,-1,-1,-1) ||
```

```
odefin(curs-1-, 3, &jono,   sizeof jono, 5, -1,-1,-1,-1,-1,-1,-1) ||
                oexec(curs[1]) ||
    ofetch(curs[1]))

    if(curs[1][0]==4)

        curlocat(12,30);
        colrprts("No Records Selected",FOREGRND,BACKGRND);
        goto close;

    else

        errrpt(curs[0],4);
        goto close;



/*  Retrieve the remaining records */
lines = 2;
head(hdr);
while (curs[1][0] != 4)

    nlines = 0;
    strcpy(line, clno);
    strcat(line, "             ");
    strcat(line, cfno);
    strcat(line, " ");
    strcat(line, clno);
    strcat(line, " ");
    strcat(line, jono);

    if (strcmp(line, " ") != 0)

        free(jobord[nlines]);
        j = strlen(line);
        jobord[nlines] = calloc(j+1,1);
```

```
        strcpy(jobord[nlines], line);
        nlines++;


/* Check for a very large entry */
if (nlines > 21)

        clscolor(FOREGRND,BACKGRND);
        curlocat(12,24);
        colrprts("**********Entry exceeds 20 lines",
            FOREGRND,BACKGRND);
        curlocat(24,21);
        colrprts("Press any key to continue or Q to quit",
            FOREGRND,BACKGRND);
        getkey(&j);
        if ((j == 'q') || (j == 'Q')) goto done;
        clscolor(FOREGRND,BACKGRND);
        lines = 2;
        for (j=0; j < nlines; j++) printf("          %s n",jobord[j]);


/* Check for a full screen */
else if (lines + nlines > 23)

        curlocat(24,21);
        colrprts("Press any key to continue or Q to quit",
            FOREGRND,BACKGRND);
        getkey(&j);
        lines = 2;
        if ((j == 'q') || (j == 'Q')) goto done;
        clscolor(FOREGRND,BACKGRND);
        head(hdr);


/* Check for a full page */
```

```
if (pflag && ((plines + nlines) > 51))

    plines = 0;
    i = (80 - strlen(hdr))/2;
    lprtff();
    for (j=1; j <= 6; j++) lprtlf();
    for (j=1; j <= i; j++) lprtchar(0, ' ');
    lprtstr(hdr);
    lprtcr();
    lprtlf();
    lprtlf();

/* Print the lines */
for(j=0; j < nlines; j++)

    curlocat(lines,10);
    colrprts(jobord[j],FOREGRND,BACKGRND);
    if (pflag)

        for (j=1; j <= 9; j++) lprtchar(0, ' ');
        lprtstr(jobord[j]);
        lprtcr();
        lprtlf();
        plines++;

    lines++;

lines++;
if (pflag)

    lprtcr();
    lprtlf();
    plines++;
```

```
ofetch(curs[1]);


  close:
  curlocat(24,20);
  colrprts("        Press a key to continue          ",
                FOREGRND,BACKGRND);
  pause();

  done:

  /*  Close the jobord cursor */
  oclose(curs[1]); /*  Free the jobord array */
  for(i=0; i < 50; i++) free(jobord[i]);
  if (pflag) lprtff();
  setscmod(EIGHTY);
  border(BACKGRND);
  clscolor(FOREGRND,BACKGRND);



/*************************************************************/
/*                  program module gethour                   */
/*                      version 1.0                          */
/*             authors:  Richard N. Woodman                  */
/*                       Michael F Rall                      */
/*                                                           */
/*                                                           */
/*                                                           */
/*          Program last modified 20 January 1986           */
/*                                                           */
/*  Purpose:  Displays budget vs expense by cost function    */
/*  cost class for hours to date.                            */
/*                                                           */
/*  Other modules called:  SELHOUR,SELEHOUR                  */
```

```
/*                                                              */
/*  Called by: INDVDISP                                         */
/*                                                              */
/*                                                              */
/*  Files used:  NONE                                           */
/*                                                              */
/*                                                              */
/*  Local variables: cfno,clno,bud,exp                          */
/*      line-80-, *budget-100-, *calloc();                      */
/*      i, j, pg, pflag, plines, lines, nlines;                 */
/*                                                              */
/**************************************************************/

gethour(select, hdr, curs)
   char *select,
          *hdr;
   short curs[][32];


   char
        cfno(5),
        clno(5),
        bud(30),
        exp(30);

   char line(80), *budget(100), *calloc();
   int  i, j, pg, pflag, plines, lines, nlines;


      nlines = 0;

   setscmod(EIGHTY);
   clscolor(FOREGRND,BACKGRND);
   border(BACKGRND);
   curlocat(12,23);
   for(i=0; i < 50; i++) budget[i] = calloc(1,1);
   colrprts("Do you want printed output (Y/N)?",FOREGRND,BACKGRND);
```

```
if (getyesno(1))

    /*  Initialize the print variables */
    pflag = 1;
    for (j=1; j <= 6; j++) lprtlf();
    for (j=1; j <= 1; j++) lprtchar(0, ' ');
    lprtstr(hdr);
    lprtcr();
    lprtlf();
    lprtlf();

else pflag = 0;
plines = 0;

  clscolor(FOREGRND,BACKGRND);
/* Process the ORACLE request */
/*  Open a cursor for the budget */
if (oopen(curs[1],curs[0],-1,-1,-1,-1,-1))

    errrpt(curs[0],4);
    goto close;


/*  Retrieve the first record */
   /* SELECT COST_FUN_NO, COST_CL_NO, HOURS
   FROM BUDGET    */

if (osql3(curs[1], select, -1) ||
odefin(curs-1-, 1, &cfno,  sizeof cfno, 5, -1,-1,-1,-1,-1,-1,-1) ||
odefin(curs-1-, 2, &clno,  sizeof clno,5, -1,-1,-1,-1,-1,-1,-1) ||
odefin(curs-1-, 3, C.3,   sizeof bud, 5, -1,-1,-1,-1,-1,-1,-1) ||
                oexec(curs[1]) ||
    ofetch(curs[1]))

    if(curs[1][0]==4)
```

```
            curlocat(12,30);
            colrprts("No Records Selected",FOREGRND,BACKGRND);
            goto close;

        else

            errrpt(curs[0],4);
            goto close;


/* Open a cursor for the expense */
/*  SELECT COST_FUN_NO, COST_CL_NO, HOURS
        FROM EXPENSE WHERE DT = (SELECT MAX(DT) FROM EXPENSE) AND
    COST_FUN_NO = :CFNO AND COST_CL_NO = :CLNO */
if (oopen(curs[2],curs[0],-1,-1,-1,-1,-1) ||
    osql3(curs[2], selehour, -1) ||
    odefin(curs[2], 1, &cfno,    sizeof cfno,
       5, -1,-1,-1,-1,-1,-1,-1) ||
    odefin(curs[2], 2, &clno,    sizeof clno,
       5, -1,-1,-1,-1,-1,-1,-1) ||
    odefin(curs[2], 3, &exp,     sizeof exp,
       5, -1,-1,-1,-1,-1,-1,-1) ||
    obndrv(curs[2],":CFNO",-1,&cfno,-1,1,-1,-1,-1,-1) ||
    obndrv(curs[2],":CLNO",-1,&clno,-1,1,-1,-1,-1,-1))

    errrpt(curs[0],4);
    goto close;


/*  Retrieve the remaining records */
lines = 2;
head(hdr);
while (curs[1][0] != 4)

    nlines = 0;
```

```
strcpy(line, cfno);
strcat(line, "   ");
strcat(line, clno);
strcat(line, " ");
strcat(line, bud);


/*  Retrieve the first address record */
if (oexec(curs[2]) ||
    ofetch(curs[2]))

    if(curs[2][0]==4) ;
    else

        errrpt(curs[0],4);
        goto close;


while (curs[2][0] != 4)

    strcat(line, "  ");
    strcat(line, exp);
    if (strcmp(line, " ") != 0)

        free(budget[nlines]);
        j = strlen(line);
        budget[nlines] = calloc(j+1,1);
        strcpy(budget[nlines], line);
        nlines++;


    ofetch(curs[2]);


/* Check for a very large entry */
if (nlines > 21)
```

```
        clscolor(FOREGRND,BACKGRND);
        curlocat(12,24);
        colrprts("**********Entry exceeds 20 lines",
             FOREGRND,BACKGRND);
        curlocat(24,21);
        colrprts("Press any key to continue or Q to quit",
             FOREGRND,BACKGRND);
        getKey(&j);
        if ((j == 'q') || (j == 'Q')) goto done;
        clscolor(FOREGRND,BACKGRND);
        lines = 2;
        for (j=0; j < nlines; j++) printf("          %s n",budget[j]);


    /* Check for a full screen */
    else if (lines + nlines > 23)

        curlocat(24,21);
        colrprts("Press any key to continue or Q to quit",
             FOREGRND,BACKGRND);
        getkey(&j);
        lines = 2;
        if ((j == 'q') || (j == 'Q')) goto done;
        clscolor(FOREGRND,BACKGRND);
        head(hdr);


    /* Check for a full page */
    if (pflag && ((plines + nlines) > 51))

        plines = 0;
        i = (80 - strlen(hdr))/2;
        lprtff();
        for (j=1; j <= 6; j++) lprtlf();
```

```
            for (j=1; j <= i; j++) lprtchar(0, ' ');
            lprtstr(hdr);
            lprtcr();
            lprtlf();
            lprtlf();

        /* Print the lines */
        for(j=0; j < nlines; j++)

            curlocat(lines,10);
            colrprts(budget[j],FOREGRND,BACKGRND);
            if (pflag)

                for (j=1; j <= 9; j++) lprtchar(0, ' ');
                lprtstr(budget[j]);
                lprtcr();
                lprtlf();
                plines++;

            lines++;

        lines++;
        if (pflag)

            lprtcr();
            lprtlf();
            plines++;

ofetch(curs[1]);

  close:
  curlocat(24,20);
  colrprts("        Press a Key to continue          ",
```

```
                    FOREGRND,BACKGRND);
  pause();

  done:

  /*  Close the budget cursor */
  oclose(curs[1]);
  /*  Free the budget array */
  for(i=0; i < 50; i++) free(budget[i]);
  if (pflag) lprtff();
  setscmod(EIGHTY);
  border(BACKGRND);
  clscolor(FOREGRND,BACKGRND);
```

```
/*************************************************************/
/*                  program module getlab                    */
/*                      version 1.0                          */
/*             authors:  Richard N. Woodman                  */
/*                       Michael F Rall                      */
/*                                                           */
/*                                                           */
/*                                                           */
/*          Program last modified 20 January 1986           */
/*                                                           */
/*  Purpose:  Displays budget vs expense by cost function    */
/*  cost class for labor to date.                            */
/*                                                           */
/*  Other modules called:  SELLAB, SELELAB                   */
/*                                                           */
/*  Called by: INDVDISP                                      */
/*                                                           */
/*                                                           */
```

```
/*  Files used:  NONE                                          */
/*                                                             */
/*                                                             */
/*  Local variables: cfno,clno,bud,exp                         */
/*      line-80-, *budget-100-, *calloc( );                    */
/*      i, j, pg, pflag, plines, lines, nlines;                */
/*                                                             */
/*************************************************************/


getlab(select, hdr, curs)
   char *select,
          *hdr;
   short curs( )(32);


   char
       cfno(5),
       clno(5),
       bud(20),
       exp(20);

   char line(80), *budget(100), *calloc( );
   int  i, j, pg, pflag, plines, lines, nlines;


      nlines = 0;

   setscmod(EIGHTY);
   clscolor(FOREGRND,BACKGRND);
   border(BACKGRND);
   curlocat(12,23);
   for(i=0; i < 50; i++) budget[i] = calloc(1,1);
   colrprts("Do you want printed output (Y/N)?",FOREGRND,BACKGRND);
   if (getyesno(1))

       /*  Initialize the print variables */
```

```
    pflag = 1;
    for (j=1; j <= 6; j++) lprtlf();
    for (j=1; j <= 1; j++) lprtchar(0, ' ');
    lprtstr(hdr);
    lprtcr();
    lprtlf();
    lprtlf();

else pflag = 0;
plines = 0;

 clscolor(FOREGRND,BACKGRND);
/* Process the ORACLE request */
/*  Open a cursor for the budget */
if (oopen(curs[1],curs[0],-1,-1,-1,-1,-1))

    errrpt(curs[0],4);
    goto close;

/*  Retrieve the first record */
   /* SELECT COST_FUN_NO, COST_CL_NO, LABOR
   FROM BUDGET    */

if (osql3(curs[1], select, -1) ||
odefin(curs-1-, 1, &cfno,  sizeof cfno, 5, -1,-1,-1,-1,-1,-1,-1) ||
odefin(curs-1-, 2, &clno,  sizeof clno, 5, -1,-1,-1,-1,-1,-1,-1) ||
odefin(curs-1-, 3, C.3,   sizeof bud, 5, -1,-1,-1,-1,-1,-1,-1) ||
                oexec(curs[1]) ||
    ofetch(curs[1]))

    if(curs[1][0]==4)

        curlocat(12,30);
        colrprts("No Records Selected",FOREGRND,BACKGRND);
        goto close;
```

```
    else

        errrpt(curs[0],4);
        goto close;


/* Open a cursor for the expense */
/*  SELECT COST_FUN_NO, COST_CL_NO, LABOR
       FROM EXPENSE WHERE DT = (SELECT MAX(DT) FROM EXPENSE) AND
    COST_FUN_NO = :CFNO AND COST_CL_NO = :CLNO */
if (oopen(curs[2],curs[0],-1,-1,-1,-1,-1) ||
    osql3(curs[2], selelab, -1) ||
    odefin(curs[2], 1, &cfno,    sizeof cfno,
       5, -1,-1,-1,-1,-1,-1,-1) ||
    odefin(curs[2], 2, &clno,    sizeof clno,
       5, -1,-1,-1,-1,-1,-1,-1) ||
    odefin(curs[2], 3, &exp,     sizeof exp,
       5, -1,-1,-1,-1,-1,-1,-1) ||
    obndrv(curs[2],":CFNO",-1,&cfno,-1,1,-1,-1,-1,-1) ||
    obndrv(curs[2],":CLNO",-1,&clno,-1,1,-1,-1,-1,-1))

    errrpt(curs[0],4);
    goto close;


/*  Retrieve the remaining records */
lines = 2;
head(hdr);
while (curs[1][0] != 4)

    nlines = 0;
    strcpy(line, cfno);
    strcat(line, "       ");
    strcat(line, clno);
```

```
strcat(line, " ");
strcat(line, bud);


/*  Retrieve the first address record */
if (oexec(curs[2]) ||
    ofetch(curs[2]))

    if(curs[2][0]==4) ;
    else

        errrpt(curs[0],4);
        goto close;


while (curs[2][0] != 4)

    strcat(line, "  ");
    strcat(line, exp);
    if (strcmp(line, " ") != 0)

        free(budget[nlines]);
        j = strlen(line);
        budget[nlines] = calloc(j+1,1);
        strcpy(budget[nlines], line);
        nlines++;


    ofetch(curs[2]);


/* Check for a very large entry */
if (nlines > 21)

    clscolor(FOREGRND,BACKGRND);
    curlocat(12,24);
```

```
    colrprts("**********Entry exceeds 20 lines",
        FOREGRND,BACKGRND);
    curlocat(24,21);
    colrprts("Press any key to continue or Q to quit",
        FOREGRND,BACKGRND);
    getkey(&j);
    if ((j == 'q') || (j == 'Q')) goto done;
    clscolor(FOREGRND,BACKGRND);
    lines = 2;
    for (j=0; j < nlines; j++) printf("          %s n",budget[j]);
```

#

```
/* Check for a full screen */
else if (lines + nlines > 23)

    curlocat(24,21);
    colrprts("Press any key to continue or Q to quit",
        FOREGRND,BACKGRND);
    getkey(&j);
    lines = 2;
    if ((j == 'q') || (j == 'Q')) goto done;
    clscolor(FOREGRND,BACKGRND);
    head(hdr);

/* Check for a full page */
if (pflag && ((plines + nlines) > 51))

    plines = 0;
    i = (80 - strlen(hdr))/2;
    lprtff();
    for (j=1; j <= 6; j++) lprtlf();
    for (j=1; j <= i; j++) lprtchar(0, ' ');
    lprtstr(hdr);
    lprtcr();
```

```
            lprtlf();
            lprtlf();

        /* Print the lines */
        for(j=0; j < nlines; j++)

            curlocat(lines,10);
            colrprts(budget[j],FOREGRND,BACKGRND);
            if (pflag)

                for (j=1; j <= 9; j++) lprtchar(0, ' ');
                lprtstr(budget[j]);
                lprtcr();
                lprtlf();
                plines++;

            lines++;

        lines++;
        if (pflag)

            lprtcr();
            lprtlf();
            plines++;

ofetch(curs[1]);

  close:
  curlocat(24,20);
  colrprts("        Press a Key to continue          ",
                    FOREGRND,BACKGRND);
  pause();

  done:
```

```
   /*  Close the budget cursor */
   oclose(curs[1]);
   /*  Free the budget array */
   for(i=0; i < 50; i++) free(budget[i]);
   if (pflag) lprtff();
   setscmod(EIGHTY);
   border(BACKGRND);
   clscolor(FOREGRND,BACKGRND);



/*************************************************************/
/*                  program module getmat                    */
/*                      version 1.0                          */
/*             authors:  Richard N. Woodman                  */
/*                       Michael F Rall                      */
/*                                                           */
/*                                                           */
/*                                                           */
/*          Program last modified 20 January 1986           */
/*                                                           */
/*  Purpose:  Displays budget vs expense by cost function    */
/*  cost class for material to date.                          */
/*                                                           */
/*  Other modules called:  SELMAT, SELEMAT                   */
/*                                                           */
/*  Called by: INDVDISP                                      */
/*                                                           */
/*                                                           */
/*  Files used:  NONE                                        */
/*                                                           */
/*                                                           */
/*  Local variables: cfno,clno,bud,exp                       */
/*      line-80-, *budget-100-, *calloc( );                  */
```

```
/*      i, j, pg, pflag, plines, lines, nlines;          */
/*                                                       */
/*******************************************************/

getmat(select, hdr, curs)
   char *select,
          *hdr;
   short curs[][32];


   char
        efno[5],
        clno[5],
        bud[20],
        exp[20];

   char line[80], *budget[100], *calloc();
   int  i, j, pg, pflag, plines, lines, nlines;


      nlines = 0;

   setscmod(EIGHTY);
   clscolor(FOREGRND,BACKGRND);
   border(BACKGRND);
   curlocat(12,23);
   for(i=0; i < 50; i++) budget[i] = calloc(1,1);
   colrprts("Do you want printed output (Y/N)?",FOREGRND,BACKGRND);
   if (getyesno(1))

        /*  Initialize the print variables */
        pflag = 1;
        for (j=1; j <= 6; j++) lprtlf();
        for (j=1; j <= 1; j++) lprtchar(0, ' ');
        lprtstr(hdr);
        lprtcr();
        lprtlf();
```

```
    lprtlf();

else pflag = 0;
plines = 0;

 clscolor(FOREGRND,BACKGRND);
/* Process the ORACLE request */
/*  Open a cursor for the budget */
if (oopen(curs[1],curs[0],-1,-1,-1,-1,-1))

    errrpt(curs[0],4);
    goto close;


/*  Retrieve the first record */
   /* SELECT COST_FUN_NO, COST_CL_NO, MATERIAL
   FROM BUDGET    */

if (osql3(curs[1], select, -1) ||
odefin(curs-1-, 1, &cfno,  sizeof cfno, 5, -1,-1,-1,-1,-1,-1,-1) ||
odefin(curs-1-, 2, &clno,  sizeof clno, 5, -1,-1,-1,-1,-1,-1,-1) ||
odefin(curs[1], 3, C.3,   sizeof bud,   5, -1,-1,-1,-1,-1,-1,-1) ||
                oexec(curs[1]) ||
    ofetch(curs[1]))

    if(curs[1][0]==4)

        curlocat(12,30);
        colrprts("No Records Selected",FOREGRND,BACKGRND);
        goto close;

    else

        errrpt(curs[0],4);
        goto close;
```

```
/* Open a cursor for the expense */
/*  SELECT COST_FUN_NO, COST_CL_NO, MATERIAL
       FROM EXPENSE WHERE  DT = (SELECT MAX(DT) FROM EXPENSE) AND
    COST_FUN_NO = :CFNO AND COST_CL_NO = :CLNO */
if (oopen(curs[2],curs[0],-1,-1,-1,-1,-1) ||
    osql3(curs[2], selemat, -1) ||
    odefin(curs[2], 1, &cfno,    sizeof cfno,
       5, -1,-1,-1,-1,-1,-1,-1) ||
    odefin(curs[2], 2, &clno,    sizeof clno,
       5, -1,-1,-1,-1,-1,-1,-1) ||
    odefin(curs[2], 3, &exp,     sizeof exp,
       5, -1,-1,-1,-1,-1,-1,-1) ||
    obndrv(curs[2],":CFNO",-1,&cfno,-1,1,-1,-1,-1,-1) ||
    obndrv(curs[2],":CLNO",-1,&clno,-1,1,-1,-1,-1,-1))

    errrpt(curs[0],4);
    goto close;

/*  Retrieve the remaining records */
lines = 2;
head(hdr);
while (curs[1][0] != 4)

    nlines = 0;
    strcpy(line, cfno);
    strcat(line, "       ");
    strcat(line, clno);
    strcat(line, " ");
    strcat(line, bud);

    /*  Retrieve the first address record */
    if (oexec(curs[2]) ||
        ofetch(curs[2]))
```

```
    if(curs[2][0]==4) ;
    else

        errrpt(curs[0],4);
        goto close;


while (curs[2][0] != 4)

    strcat(line, "  ");
    strcat(line, exp);
    if (strcmp(line, " ") != 0)

        free(budget[nlines]);
        j = strlen(line);
        budget[nlines] = calloc(j+1,1);
        strcpy(budget[nlines], line);
        nlines++;


    ofetch(curs[2]);

/* Check for a very large entry */
if (nlines > 21)

    clscolor(FOREGRND,BACKGRND);
    curlocat(12,24);
    colrprts("**********Entry exceeds 20 lines",
        FOREGRND,BACKGRND);
    curlocat(24,21);
    colrprts("Press any key to continue or Q to quit",
        FOREGRND,BACKGRND);
    getkey(&j);
    if ((j == 'q') || (j == 'Q')) goto done;
```

```
        clscolor(FOREGRND,BACKGRND);
        lines = 2;
        for (j=0; j < nlines; j++) printf("          %s n",budget[j]);



/* Check for a full screen */
else if (lines + nlines > 23)

        curlocat(24,21);
        colrprts("Press any key to continue or Q to quit",
             FOREGRND,BACKGRND);
        getkey(&j);
        lines = 2;
        if ((j == 'q') || (j == 'Q')) goto done;
        clscolor(FOREGRND,BACKGRND);
        head(hdr);


/* Check for a full page */
if (pflag && ((plines + nlines) > 51))

        plines = 0;
        i = (80 - strlen(hdr))/2;
        lprtff();
        for (j=1; j <= 6; j++) lprtlf();
        for (j=1; j <= i; j++) lprtchar(0, ' ');
        lprtstr(hdr);
        lprtcr();
        lprtlf();
        lprtlf();


/* Print the lines */
for(j=0; j < nlines; j++)
```

```
            curlocat(lines,10);
            colrprts(budget[j],FOREGRND,BACKGRND);
            if (pflag)

                for (j=1; j <= 9; j++) lprtchar(0, ' ');
                lprtstr(budget[j]);
                lprtcr();
                lprtlf();
                plines++;

            lines++;


        lines++;
        if (pflag)

            lprtcr();
            lprtlf();
            plines++;

ofetch(curs[1]);


  close:
  curlocat(24,20);
  colrprts("        Press a key to continue          ",
                    FOREGRND,BACKGRND);
  pause();

  done:

  /*  Close the budget cursor */
  oclose(curs[1]);
  /*  Free the budget array */
  for(i=0; i < 50; i++) free(budget[i]);
  if (pflag) lprtff();
  setscmod(EIGHTY);
```

```
   border(BACKGRND);
   clscolor(FOREGRND,BACKGRND);



/*************************************************************/
/*                    program module getoth                  */
/*                        version 1.0                        */
/*              authors:  Richard N. Woodman                 */
/*                        Michael F Rall                     */
/*                                                           */
/*                                                           */
/*                                                           */
/*           Program last modified 20 January 1986          */
/*                                                           */
/*  Purpose:  Displays budget vs expense by cost function    */
/*  cost class for other to date.                            */
/*                                                           */
/*  Other modules called:  SELOTH, SELEOTH                   */
/*                                                           */
/*  Called by: INDVDISP                                      */
/*                                                           */
/*                                                           */
/*  Files used:  NONE                                        */
/*                                                           */
/*                                                           */
/*  Local variables: cfno,clno,bud,exp                       */
/*      line-80-, *budget-100-, *calloc( );                  */
/*      i, j, pg, pflag, plines, lines, nlines;              */
/*                                                           */
/*************************************************************/

getoth(select, hdr, curs)
   char *select,
        *hdr;
```

```
short curs[][32];


char
     cfno[5],
     clno[5],
     bud[20],
     exp[20];

char line[80],  *budget[100], *calloc();
int  i, j, pg, pflag, plines, lines, nlines;


   nlines = 0;

setscmod(EIGHTY);
clscolor(FOREGRND,BACKGRND);
border(BACKGRND);
curlocat(12,23);
for(i=0; i < 50; i++) budget[i] = calloc(1,1);
colrprts("Do you want printed output (Y/N)?",FOREGRND,BACKGRND);
if (getyesno(1))

     /*  Initialize the print variables */
     pflag = 1;
     for (j=1; j <= 6; j++) lprtlf();
     for (j=1; j <= 1; j++) lprtchar(0, ' ');
     lprtstr(hdr);
     lprtcr();
     lprtlf();
     lprtlf();

else pflag = 0;
plines = 0;

  clscolor(FOREGRND,BACKGRND);
/* Process the ORACLE request */
```

```
/*  Open a cursor for the budget */
if (oopen(curs[1],curs[0],-1,-1,-1,-1,-1))

     errrpt(curs[0],4);
     goto close;

/*  Retrieve the first record */
   /* SELECT COST_FUN_NO, COST =_CL_NO, OTHER
   FROM BUDGET */

if (osql3(curs[1], select, -1) ||
odefin(curs-1-, 1, &cfno,  sizeof cfno, 5, -1,-1,-1,-1,-1,-1,-1) ||
odefin(curs-1-, 2, &clno,  sizeof clno, 5, -1,-1,-1,-1,-1,-1,-1) ||
odefin(curs-1-, 3, C.3,   sizeof bud, 5, -1,-1,-1,-1,-1,-1,-1) ||
                oexec(curs[1]) ||
     ofetch(curs[1]))

     if(curs[1][0]==4)

          curlocat(12,30);
          colrprts("No Records Selected",FOREGRND,BACKGRND);
          goto close;

     else

          errrpt(curs[0],4);
          goto close;

/* Open a cursor for the expense */
/*  SELECT COST_FUN_NO, COST_CL_NO, OTHER
        FROM EXPENSE WHERE DT = (SELECT MAX(DT) FROM EXPENSE) AND
     COST_FUN_NO = :CFNO AND COST_CL_NO = :CLNO */
if (oopen(curs[2],curs[0],-1,-1,-1,-1,-1) ||
     osql3(curs[2], seleoth, -1) ||
```

```
    odefin(curs[2], 1, &cfno,    sizeof cfno,
       5, -1,-1,-1,-1,-1,-1,-1) ||
    odefin(curs[2], 2, &clno,    sizeof clno,
       5, -1,-1,-1,-1,-1,-1,-1) ||
    odefin(curs[2], 3, &exp,     sizeof exp,
       5, -1,-1,-1,-1,-1,-1,-1) ||
    obndrv(curs[2],":CFNO",-1,&cfno,-1,1,-1,-1,-1,-1) ||
    obndrv(curs[2],":CLNO",-1,&clno,-1,1,-1,-1,-1,-1))

    errrpt(curs[0],4);
    goto close;

/*  Retrieve the remaining records */
lines = 2;
head(hdr);
while (curs[1][0] != 4)

    nlines = 0;
    strcpy(line, cfno);
    strcat(line, "       ");
    strcat(line, clno);
    strcat(line, " ");
    strcat(line, bud);

    /*  Retrieve the first address record */
    if (oexec(curs[2]) ||
        ofetch(curs[2]))

        if(curs[2][0]==4) ;
        else

            errrpt(curs[0],4);
            goto close;
```

```
while (curs[2][0] != 4)

    strcat(line, "  ");
    strcat(line, exp);
    if (strcmp(line, " ") != 0)

        free(budget[nlines]);
        j = strlen(line);
        budget[nlines] = calloc(j+1,1);
        strcpy(budget[nlines], line);
        nlines++;

    ofetch(curs[2]);

/* Check for a very large entry */
if (nlines > 21)

    clscolor(FOREGRND,BACKGRND);
    curlocat(12,24);
    colrprts("**********Entry exceeds 20 lines",
        FOREGRND,BACKGRND);
    curlocat(24,21);
    colrprts("Press any key to continue or Q to quit",
        FOREGRND,BACKGRND);
    getkey(&j);
    if ((j == 'q') || (j == 'Q')) goto done;
    clscolor(FOREGRND,BACKGRND);
    lines = 2;
    for (j=0; j < nlines; j++) printf("         %s n",budget[j]);


/* Check for a full screen */
```

```
else if (lines + nlines > 23)

     curlocat(24,21);
     colrprts("Press any key to continue or Q to quit",
          FOREGRND,BACKGRND);
     getkey(&j);
     lines = 2;
     if ((j == 'q') || (j == 'Q')) goto done;
     clscolor(FOREGRND,BACKGRND);
     head(hdr);


/* Check for a full page */
if (pflag && ((plines + nlines) > 51))

     plines = 0;
     i = (80 - strlen(hdr))/2;
     lprtff();
     for (j=1; j <= 6; j++) lprtlf();
     for (j=1; j <= i; j++) lprtchar(0, ' ');
     lprtstr(hdr);
     lprtcr();
     lprtlf();
     lprtlf();


/* Print the lines */
for(j=0; j < nlines; j++)

     curlocat(lines,10);
     colrprts(budget[j],FOREGRND,BACKGRND);
     if (pflag)

          for (j=1; j <= 9; j++) lprtchar(0, ' ');
          lprtstr(budget[j]);
          lprtcr();
```

```
                lprtlf( );
                plines++;

            lines++;

        lines++;
        if (pflag)

            lprtcr( );
            lprtlf( );
            plines++;

ofetch(curs[1]);

  close:
  curlocat(24,20);
  colrprts("        Press a key to continue          ",
                    FOREGRND,BACKGRND);
  pause( );

  done:

  /*  Close the budget cursor */
  oclose(curs[1]);
  /*  Free the budget array */
  for(i=0; i < 50; i++) free(budget[i]);
  if (pflag) lprtff( );
  setscmod(EIGHTY);
  border(BACKGRND);
  clscolor(FOREGRND,BACKGRND);
```

```
/*************************************************************/
```

```
/*                  program module getsum                    */
/*                      version 1.0                          */
/*              authors:  Richard N. Woodman                 */
/*                        Michael F Rall                     */
/*                                                           */
/*                                                           */
/*                                                           */
/*           Program last modified 20 January 1986          */
/*                                                           */
/*  Purpose:  Displays budget vs expense total               */
/*                        to date.                           */
/*                                                           */
/*  Other modules called:  SELSUM, SELSUMA                   */
/*                                                           */
/*  Called by: TOBUDEXP                                      */
/*                                                           */
/*                                                           */
/*  Files used:  NONE                                        */
/*  Files created :  bud,graf1                               */
/*                                                           */
/*  Local variables: cfno,dte, bud,exp                       */
/*      line-80-, *budget-100-, *calloc( );                  */
/*     i, j, pg, pflag, plines, lines, nlines;               */
/*                                                           */
/*************************************************************/

 getsum(select, hdr, curs)
   char *select,
          *hdr;
   short curs[ ][32];


   char
       cfno[5],
       dte[10],
       bud[15],
```

```
    exp[15];

char line[80], *budget[100], *calloc();
int  i, j, pg, pflag, gflag, plines, lines, nlines;


   nlines = 0;

setscmod(EIGHTY);
clscolor(FOREGRND,BACKGRND);
border(BACKGRND);
curlocat(12,23);
for(i=0; i < 50; i++) budget[i] = calloc(1,1);
colrprts("Do you want printed output (Y/N)?",FOREGRND,BACKGRND);
if (getyesno(1))

    /*  Initialize the print variables */
    pflag = 1;
    for (j=1; j <= 6; j++) lprtlf();
    for (j=1; j <= 1; j++) lprtchar(0, ' ');
    lprtstr(hdr);
    lprtcr();
    lprtlf();
    lprtlf();

else pflag = 0;
plines = 0;

/*  Initialize the graph variable */
curlocat(14,23);
colrprts("Graph Output (Y/N)?",FOREGRND,BACKGRND);
if (getyesno(1)) gflag = 1;
else gflag = 0;

 clscolor(FOREGRND,BACKGRND);
/* Process the ORACLE request */
/*  Open a cursor for the budget */
```

```
if (oopen(curs[1],curs[0],-1,-1,-1,-1,-1))

     errrpt(curs[0],4);
     goto close;


/*  Retrieve the first record */
   /* SELECT SUM(LABOR)+SUM(MATERIAL)+SUM(OTHER)
   FROM BUDGET WHERE COST_FUN_NO < '9200' */

if (osql3(curs[1], select, -1) ||
     odefin(curs-1-, 1, C.3,   sizeof bud, 5, -1,-1,-1,-1,-1,-1,-1) ||
                oexec(curs[1]) ||
     ofetch(curs[1]))

     if(curs[1][0]==4)

          curlocat(12,30);
          colrprts("No Records Selected",FOREGRND,BACKGRND);
          goto close;

     else

          errrpt(curs[0],4);
          goto close;


/* Open a cursor for the expense */
/* SELECT  SUM(LABOR)+SUM(MATERIAL)+SUM(OTHER), DT
   FROM EXPENSE
   WHERE COST_FUN_NO = :COST_FUN_NO GROUP BY DT*/
if (oopen(curs[2],curs[0],-1,-1,-1,-1,-1) ||
     osql3(curs[2], selsuma, -1) ||
     odefin(curs[2], 1, &exp,     sizeof exp,
        5, -1,-1,-1,-1,-1,-1,-1) ||
     odefin(curs[2], 2, &dte,     sizeof dte,
```

```
        5, -1,-1,-1,-1,-1,-1,-1) )

    errrpt(curs[0],4);
    goto close;

/*  Retrieve the remaining records */
lines = 2;
head(hdr);
while (curs[1][0] != 4)

    nlines = 0;
    strcpy(line, bud);

        if (strcmp(line, " ") != 0)

            free(budget[nlines]);
            j = strlen(line);
            budget[nlines] = calloc(j+1,1);
            strcpy(budget[nlines], line);
            if (gflag == 1) writefb(line);
            nlines++;


    /*  Retrieve the first expense record */
    if (oexec(curs[2]) ||
        ofetch(curs[2]))

        if(curs[2][0]==4) ;
        else

            errrpt(curs[0],4);
            goto close;
```

```
while (curs[2][0] != 4)

    strcpy(line, "                ");
    strcat(line, exp);
    strcat(line, "          ");
    strcat(line, dte);
    if (strcmp(line, " ") != 0)

        free(budget[nlines]);
        j = strlen(line);
        budget[nlines] = calloc(j+1,1);
        strcpy(budget[nlines], line);
        if (gflag == 1) writefl(line);
        nlines++;


    ofetch(curs[2]);


/* Check for a very large entry */
if (nlines > 21)

    clscolor(FOREGRND,BACKGRND);
    curlocat(12,24);
    colrprts("**********Entry exceeds 20 lines",
        FOREGRND,BACKGRND);
    curlocat(24,21);
    colrprts("Press any key to continue or Q to quit",
        FOREGRND,BACKGRND);
    getKey(&j);
    if ((j == 'q') || (j == 'Q')) goto done;
    clscolor(FOREGRND,BACKGRND);
    lines = 2;
    for (j=0; j < nlines; j++) printf("          %s n",budget[j]);
```

```
/* Check for a full screen */
else if (lines + nlines > 23)

     curlocat(24,21);
     colrprts("Press any key to continue or Q to quit",
          FOREGRND,BACKGRND);
     getkey(&j);
     lines = 2;
     if ((j == 'q') || (j == 'Q')) goto done;
     clscolor(FOREGRND,BACKGRND);
     head(hdr);


/* Check for a full page */
if (pflag && ((plines + nlines) > 51))

     plines = 0;
     i = (80 - strlen(hdr))/2;
     lprtff();
     for (j=1; j <= 6; j++) lprtlf();
     for (j=1; j <= i; j++) lprtchar(0, ' ');
     lprtstr(hdr);
     lprtcr();
     lprtlf();
     lprtlf();


/* Print the lines */
for(j=0; j < nlines; j++)

     curlocat(lines,10);
     colrprts(budget[j],FOREGRND,BACKGRND);
     if (pflag)

          for (j=1; j <= 9; j++) lprtchar(0, ' ');
```

```
                lprtstr(budget[j]);
                lprtcr();
                lprtlf();
                plines++;

            lines++;

        lines++;
        if (pflag)

            lprtcr();
            lprtlf();
            plines++;

        if (gflag == 1)

            i = execute2("d:lyons6.exe","lyons6");

ofetch(curs[1]);

  close:
  curlocat(24,20);
  colrprts("        Press a key to continue          ",
                FOREGRND,BACKGRND);
  pause();

  done:

  /*  Close the budget cursor */
  oclose(curs[1]);
  /*  Free the budget array */
  for(i=0; i < 50; i++) free(budget[i]);
  if (pflag) lprtff();
  setscmod(EIGHTY);
```

```
    border(BACKGRND);
    clscolor(FOREGRND,BACKGRND);
```

3. ORCAINP

```
static char selfun[ ]= "SELECT COST_CL_NO, OTLABOR+STLABOR,
                                MATERIAL, OTHER
                                FROM BUDGET WHERE COST_FUN_NO = " ;

static char selsum[ ]= "SELECT SUM(OTLABOR)+SUM(STLABOR)+
                                SUM(MATERIAL)+SUM(OTHER)
                                FROM BUDGET  " ;

static char selsuma[ ]= "SELECT  SUM(STLABOR)+SUM(OTLABOR)+
                                 SUM(MATERIAL)+SUM(OTHER),DT
                                 FROM EXPENSE GROUP BY DT " ;

static char selcls[ ]= "SELECT SUM(OTLABOR)+SUM(STLABOR)+
                                SUM(MATERIAL)+SUM(OTHER)
                                FROM BUDGET WHERE COST_CL_NO = " ;

static char selfune[ ]= "SELECT SUM(OTLABOR)+SUM(STLABOR)+
                                 SUM(MATERIAL)+SUM(OTHER)
                                 FROM EXPENSE WHERE COST_FUN_NO = " ;

static char selclse[ ]= "SELECT SUM(OTLABOR)+SUM(STLABOR)+
                                 SUM(MATERIAL)+SUM(OTHER)
                                 FROM EXPENSE WHERE COST_CL_NO = " ;

static char seltote[ ]= "SELECT SUM(OTLABOR)+SUM(STLABOR)+
                                 SUM(MATERIAL)+SUM(OTHER)
                                 FROM EXPENSE"  ;

static char selefun[ ]= "SELECT COST_FUN_NO,SUM(OTLABOR)+
                                 SUM(STLABOR)+SUM(MATERIAL)+SUM(OTHER)
            FROM EXPENSE WHERE DT = (SELECT MAX(DT) FROM EXPENSE)
            AND COST_FUN_NO = :cfno GROUP BY COST_FUN_NO" ;
```

```
static char selecl[ ]= "SELECT COST_CL_NO,SUM(OTLABOR)+
                              SUM(STLABOR)+SUM(MATERIAL)+SUM(OTHER)
          FROM EXPENSE WHERE DT = (SELECT MAX(DT) FROM EXPENSE)
          AND COST_CL_NO = :clno GROUP BY COST_CL_NO" ;

static char selbfun[ ]= "SELECT COST_FUN_NO, SUM(OTLABOR)+
                               SUM(STLABOR)+SUM(MATERIAL)+SUM(OTHER)
                               FROM BUDGET GROUP BY COST_FUN_NO" ;

static char selbcl[ ]= "SELECT COST_CL_NO, SUM(OTLABOR)+
                              SUM(STLABOR)+SUM(MATERIAL)+SUM(OTHER)
                              FROM BUDGET GROUP BY COST_CL_NO" ;

static char selbcfcl[ ]= "SELECT COST_FUN_NO, COST_CL_NO,
                                OTLABOR+STLABOR+MATERIAL+OTHER
      FROM BUDGET WHERE OTLABOR != 0 OR STLABOR != 0 OR
                                    MATERIAL != 0 OR OTHER != 0" ;

static char selecfcl[ ]= "SELECT COST_FUN_NO, COST_CL_NO,
                              OTLABOR+STLABOR+MATERIAL+OTHER FROM EXPENSE
                              WHERE DT = (SELECT MAX(DT) FROM EXPENSE) AND
      (OTLABOR != 0 OR STLABOR != 0 OR MATERIAL != 0 OR OTHER != 0)
       AND COST_FUN_NO = :CFNO AND COST_CL_NO = :CLNO" ;

static char selhour[ ]= "SELECT COST_FUN_NO, COST_CL_NO,
                                OTHOURS+STHOURS
                                FROM BUDGET  " ;

static char sellab[ ]= "SELECT COST_FUN_NO, COST_CL_NO,
                                 OTLABOR+STLABOR
                                 FROM BUDGET " ;

static char selmat[ ]= "SELECT COST_FUN_NO, COST_CL_NO, MATERIAL
                                 FROM BUDGET  " ;

static char seloth[ ]= "SELECT COST_FUN_NO, COST_CL_NO, OTHER
                               FROM BUDGET  " ;

static char selehour[ ]= "SELECT COST_FUN_NO, COST_CL_NO,
```

```
                                        OTHOURS+STHOURS
          FROM EXPENSE WHERE DT = (SELECT MAX(DT) FROM EXPENSE) AND
          COST_FUN_NO = :CFNO AND COST_CL_NO = :CLNO " ;

static char selelab[]= "SELECT COST_FUN_NO, COST_CL_NO,
                                        OTLABOR+STHOURS
          FROM EXPENSE WHERE DT = (SELECT MAX(DT) FROM EXPENSE) AND
          COST_FUN_NO = :CFNO AND COST_CL_NO = :CLNO " ;

static char selemat[]= "SELECT COST_FUN_NO, COST_CL_NO, MATERIAL
          FROM EXPENSE WHERE DT = (SELECT MAX(DT) FROM EXPENSE) AND
          COST_FUN_NO = :CFNO AND COST_CL_NO = :CLNO " ;

static char seleoth[]= "SELECT COST_FUN_NO, COST_CL_NO, OTHER
          FROM EXPENSE WHERE DT = (SELECT MAX(DT) FROM EXPENSE) AND
          COST_FUN_NO = :CFNO AND COST_CL_NO = :CLNO " ;
```

4. BAR.C

```
/*************************************************************/
/*                  program module Bar.c                     */
/*                      version 1.0                          */
/*             authors:  Richard N. Woodman                  */
/*                       Michael F Rall                      */
/*                                                           */
/*           Program last modified 20 January 1986          */
/*                                                           */
/*  This program was produced on an IBM clone using          */
/*  DOS 3.1.  Written with the C programming language,       */
/*  utilizing the GraphiC utility software.                  */
/*                                                           */
/*  This is called directly from DOS after the PROJ          */
/*  system has been processed.  This module produces a       */
/*  single bar graph, representing the budget of each cost   */
/*  center.                                                  */
/*                                                           */
/*  Files used:  GRAF                                        */
/*                                                           */
```

```
/*  External Calls:  None                                          */
/*                                                                 */
/*****************************************************************/



#include "stdio.h"
#include "graphics.h"

int _stack = 60000;

main(argc,argv)
   int argc;
   char *argv( );
                        /*  begin main  */

   FILE *infile, *outfile, *fopen( );

                                    /* Declare variables */
          char month(3), filename(30), name(4);
          float cost, x1(14),x2(14),y1(14),x(301), y(301), z(301);
          float a(301), b(14), c(301), d(301);
          int cf, flag, i, count, nxdiv, nydiv, npts;
          long gettime( );
          float  budget, budget1, budget2, z1(13), w, w1;
.pa
struct strlab   /* begin Needed for string labels. */
int flag;
char s1(10);
char s2(10);
char s3(10);
char s4(10);
char s5(10);
char s6(10);
char s7(10);
char s8(10);
char s9(10);
```

```
char s10[10];
char s11[10];
char s12[10];
char s13[10];
char s14[10];
 };                                /* end Needed for string labels. */
#if CIQ    /* Some CI86 compilers won't accept the simpler form. */
static struct strlab xstring;
xstring.flag=1;
strcpy(xstring.s1," 310112");
strcpy(xstring.s2," 310113");
strcpy(xstring.s3," 310114");
strcpy(xstring.s4," 310115");
strcpy(xstring.s5," 310116");
strcpy(xstring.s6," 310117");
strcpy(xstring.s7," 310118");
strcpy(xstring.s8," 310119");
strcpy(xstring.s9," 310112");
strcpy(xstring.s10," 310113");
#else
static struct strlab xstring =          /*  begin  */
            1,"","," 310112"," 310113"," 310114"," 310115"
                   , " 310116"," 310117"," 310118"," 310119","" ;
#endif
   strcpy(filename,"graf");
   if ((infile = fopen(filename,"r")) == NULL)

        printf("Sorry, cannot open %s", filename);
        return;

           for(i=0;i<=9;i++)

                       a[i]=0;
                       b[i]=0;
```

```
                    c[i]=0;
                    d[i]=0;

        count = 0;
.pa
  while ((flag = fscanf(infile, "%d%f%f", C.7,&cost,&budget1)) != EOF)
                                        /*  begin while         */
          a[count] = cost/1000;             /*  cost                */
          c[count] = budget1/1000;          /* cost function budget */
          d[count] = (budget1/cost)/1000;   /* cost function budget */
    count++;                              /*  increment count     */
                                        /*  end while           */

  fclose(infile);

        /* These are the strings. */
        settime()                       /* Start timing of run. */
        bgnplot(1,'g',"lyons8.tkf"); /* Initialize plot. Graphic mode */
        startplot();

                /* Change to simplex Greek and math */
        font(4,"simplex.fnt",' 310',"duplex.fnt",' 311',"complex.fnt"
                ,' 312',"simgrma.fnt",' 313');
        xlab(&xstring);                 /* Turn on string labels */

        cross(0);
        color(0);
        physor(0.0,0.0);                    /*RESET DEFAULT ORIGIN*/
        page(9.0,6.855);
        area2d(7.5,6.5);
        nxdiv=8;        /* Desired # of x-axis divisions */
        nydiv=6;        /* Desired # of y-axis divisions */
        npts=8;         /* Number of points in x and y vectors */
        for(i=0;i<count;i++) b[i]=(float)(i+1);

        color(6);
        grid(2);        /* Put fine dotted grid on plot */
```

```
        graf("",0.,1.,9.,"%-1.0f",0.,1.,10.);
        xstring.flag=0;   /* Turn off string label option */
        xlab(&xstring);   /* after axes have been drawn. */
        color(3);
        xname(" 312Cost Function");   /* Label for x-axis */
        heading(" 311BUDGET");        /* Title of plot */
        yname(" 312Millions");        /* Label for y-axis */
        color(2);
        pltfnt(12.5, 8.2, " 312110", .5, 0);

        bar(npts,b,a,npts,2);

        endplot();           /* Terminate first plot */

        stopplot();         /* Close files and quit */

             /*  end main  */
```

5. PLOT.C

```
/*************************************************************/
/*                  program module Plot.c                    */
/*                      version 1.0                          */
/*             authors:  Richard N. Woodman                  */
/*                       Michael F Rall                      */
/*                                                           */
/*           Program last modified 20 January 1986          */
/*                                                           */
/*  This program was produced on an IBM clone using          */
/*  DOS 3.1.  Written with the C programming language,       */
/*  utilizing the GraphicC utility software.                 */
/*                                                           */
/*  This is called directly from DOS after the PROJ          */
/*  system has been processed.  This module produces a       */
/*  line graph.  The solid line being the budget, with the   */
/*  broken line representing the expenses.  It is plotted    */
/*  by month.                                                */
/*                                                           */
```

```
/*  Files used:  GRAF1, BUD                                          */
/*                                                                   */
/*  External Calls:  None                                            */
/*                                                                   */
/*******************************************************************/


#include "stdio.h"
#include "graphics.h"

int _stack = 60000;

main(argc,argv)
   int argc;
   char *argv[];
                        /*  begin main  */

   FILE *infile, *outfile, *fopen();

                                  /* Declare variables */
          char month[9], filename[30], name[4];
          float cost, x1[14],x2[14],y1[14],x[301], y[301], z[301];
          float a[301], b[14], c[301], d[301];
          int counts,cf, flag, i, count, nxdiv, nydiv, npts, npt;
          long gettime();
          float  newbudget,budget, budget1, budget2, z1[13], w, w1;
.pa
struct labstra          /* begin Needed for string labels. */
int flag;
char s1[10];
char s2[10];
char s3[10];
char s4[10];
char s5[10];
char s6[10];
char s7[10];
char s8[10];
```

```
char s9[10];
char s10[10];
char s11[10];
char s12[10];
char s13[10];
char s14[10];
 };                              /* end Needed for string labels. */
#if CIQ /* Some CI86 compilers won't accept the simpler form. */
static struct labstr mstring;
nstring.flag=1;
strcpy(nstring.s1," 310OCT");
strcpy(nstring.s2," 310NOV");
strcpy(nstring.s3," 310DEC");
strcpy(nstring.s4," 310JAN");
strcpy(nstring.s5," 310FEB");
strcpy(nstring.s6," 310MAR");
strcpy(nstring.s7," 310APR");
strcpy(nstring.s8," 310MAY");
strcpy(nstring.s9," 310JUN");
strcpy(nstring.s10," 310JUL");
strcpy(nstring.s11," 310AUG");
strcpy(nstring.s12," 310SEP");
strcpy(nstring.s13," 310OCT");
strcpy(nstring.s14," 310NOV");
#else
static struct labstra nstring =          /*  begin  */
            1,"","  310OCT","  310NOV","  310DEC","  310JAN", "  310FEB"
               ,"  310MAR","  310APR","  310MAY","  310JUN","  310JUL"
               ,"  310AUG","  310SEP","" ;
#endif

   strcpy(filename,"bud");
   if ((infile = fopen(filename,"r")) == NULL)

        printf("Sorry, cannot open %s", filename);
```

```
        return;


           for(i=0;i<=1;i++)

                        x[i]=0;
                        y[i]=0;
                        z[i]=0;

.pa
    flag = fscanf(infile, "%f",&budget);
     newbudget = (budget)/1000;  /*  budget      */

           fclose(infile);

           w=0;

           for(i=1;i<=13;i++)

                        y[i]=(newbudget/12)+w;
                        w=(newbudget/12)+w;


           strcpy(filename,"grafl");
    if ((infile = fopen(filename,"r")) == NULL)

        printf("Sorry, cannot open %s", filename);
        return;


           w1=0;
           count = 1;
    while ((flag = fscanf(infile, "%f%s",&cost,&month)) != EOF)

            z[count] = (cost+w1)/1000;     /*  cost */
           w1=(cost)+w1;
     count++;    /*  increment count  */
                /*  end while              */
```

```
   fclose(infile);

/*  begin full page line plot by itself */

        settime();                      /* Start timing of run. */
        bgnplot(1,'g',"aplot.tkf");     /* Initialize plot. Graphic mode */

        startplot(0);

              /* Change to simplex Greek and math */
        font(4,"simplex.fnt",' 310',"duplex.fnt",' 311'
                ,"complex.fnt",' 312',"simgrma.fnt",' 313');
        xlab(&nstring);   /* Turn on string labels */
        cross(0);
        color(0);
        physor(0.0,0.0);      /*RESET DEFAULT ORIGIN*/
        page(9.0,6.855);
        area2d(7.5,6.0);
        box();           /* Draw a box around the plot */
        grid(2);         /* Put fine dotted grid on plot */
        fntchg(' 310'); /* Changes fonts for the axes */
        for(i=1;i<13;i++)

                                /* budget line  */
                                x[i]=(float)(i+1);

        for(i=1;i<=count;i++)

                                /* budget line  */
                                x1[i]=(float)(i+1);

        nxdiv=12;  /* Desired # of x-axis divisions */
        nydiv=5;   /* Desired # of divisions on linear axis */
        npts = 12;
   graf("",0.,1.,12.,"%-1.0f",0.,1.,10.);
        nstring.flag=0; /* Turn off string label option */
```

```
        xlab(&nstring); /* after axes have been drawn. */
        color(14);
        xname(" 310End of Month");              /* Make labels */
        yname(" 310Millions");
        heading(" 311Budget vs Expenses");
        solid();
        color(10);
        pltfnt(9.5, 9.2, " 312110", .5, 0);
        curve(x,y,npts,0); /* Draw curve with no symbols */
        chndsh();       /* Use chain-dashed line for second curve */
        color(12);
        curve(x1,z,count,0);  /* Plot second curve */
        endplot();              /* Terminate second plot */

        stopplot();

             /*  end main  */
```

6. COMBO.C

```
/*************************************************************/
/*                  program module Combo.c                   */
/*                      version 1.0                          */
/*            authors:  Richard N. Woodman                   */
/*                      Michael F Rall                       */
/*                                                           */
/*          Program last modified 20 January 1986           */
/*                                                           */
/*  This program was produced on an IBM clone using          */
/*  DOS 3.1.  Written with the C programming language,       */
/*  utilizing the GraphicC utility software.                 */
/*                                                           */
/*  This is called directly from DOS after the PROJ          */
/*  system has been processed.  This module produces a       */
/*  full page line graph, with the solid line representing   */
/*  the budget and the broKen line representing the          */
/*  expenses, by the month  Inset in the upper left hand     */
```

```
/*  corner in a single bar graph, reduced in size that     */
/*  plots the budget for each cost center.                 */
/*                                                         */
/*  Files used:  GRAF1, BUD, GRAF                          */
/*                                                         */
/*  External Calls:  None                                  */
/*                                                         */
/*********************************************************/


#include "stdio.h"
#include "graphics.h"

int _stack = 60000;

main(argc,argv)
   int argc;
   char *argv( );
                        /*  begin main  */

   FILE *infile, *outfile, *fopen( );

                                  /* Declare variables */
          char month(9), filename(30), name(4);
          float budg,newbudget,cost, x1(14),x2(14);
          float y1(14),x(301), y(301), z(301);
          float a(301), b(14), c(301), d(301);
          int counts,cf, flag, i, count, nxdiv, nydiv, npts;
          long gettime( );
          float  center,expn,budget,budget1,budget2,z1(13),w,w1;
.pa
struct strlab   /* begin Needed for string labels. */
int flag;
char s1(10);
char s2(10);
char s3(10);
char s4(10);
```

```
char s5[10];
char s6[10];
char s7[10];
char s8[10];
char s9[10];
char s10[10];
char s11[10];
char s12[10];
char s13[10];
char s14[10];
 };              /* end Needed for string labels. */
#if CIQ /* Some CI86 compilers won't accept the simpler form. */
static struct strlab xstring;
xstring.flag=1;
strcpy(xstring.s1," 310112");
strcpy(xstring.s2," 310113");
strcpy(xstring.s3," 310114");
strcpy(xstring.s4," 310115");
strcpy(xstring.s5," 310116");
strcpy(xstring.s6," 310117");
strcpy(xstring.s7," 310118");
strcpy(xstring.s8," 310119");
strcpy(xstring.s9," 310112");
strcpy(xstring.s10," 310113");
#else
static struct strlab xstring =          /*  begin  */
            1,"","," 310112"," 310113"," 310114"," 310115"
               ," 310116"," 310117"," 310118"," 310119","" ;
#endif

struct labstr   /* begin Needed for string labels. */
int flag;
char s1-10-;
char s2-10-;
char s3-10-;
```

```
char s4-10-;
char s5-10-;
char s6-10-;
char s7-10-;
char s8-10-;
char s9-10-;
char s10-10-;
char s11-10-;
char s12-10-;
char s13-10-;
char s14-10-;
 ;              /* end Needed for string labels. */
.pa
#if CIQ /* Some CI86 compilers won't accept the simpler form. */
static struct labstr mstring;
mstring.flag=1;
strcpy(mstring.s1," 310OCT");
strcpy(mstring.s2," 310NOV");
strcpy(mstring.s3," 310DEC");
strcpy(mstring.s4," 310JAN");
strcpy(mstring.s5," 310FEB");
strcpy(mstring.s6," 310MAR");
strcpy(mstring.s7," 310APR");
strcpy(mstring.s8," 310MAY");
strcpy(mstring.s9," 310JUN");
strcpy(mstring.s10," 310JUL");
strcpy(mstring.s11," 310AUG");
strcpy(mstring.s12," 310SEP");
strcpy(mstring.s13," 310OCT");
strcpy(mstring.s14," 310NOV");
#else
static struct labstr mstring =          /*  begin  */
            1,"","  310OCT","  310NOV","  310DEC","  310JAN"
               ,"  310FEB","  310MAR","  310APR","  310MAY"
               ,"  310JUN","  310JUL","  310AUG","  310SEP","" ;
```

```
#endif

   strcpy(filename,"bud");
   if ((infile = fopen(filename,"r")) == NULL)
        /*  begin if  */
        printf("Sorry, cannot open %s", filename);
        return;
        /*  end if  */

           for(i=0;i<=1;i++)

                      x[i]=0;
                      y[i]=0;
                      z[i]=0;

   flag = fscanf(infile, "%f",&budget);
    newbudget = (budget)/1000;  /*  budget      */

           fclose(infile);

           w=0;

           for(i=1;i<=13;i++)

                      y[i]=(newbudget/12)+w;
                      w=(newbudget/12)+w;

.pa
           strcpy(filename,"grafl");
   if ((infile = fopen(filename,"r")) == NULL)
                /*  begin if  */
        printf("Sorry, cannot open %s", filename);
        return;
        /*  end if  */

           w1=0;
           count = 1;
   while ((flag = fscanf(infile, "%f%s",&cost,&month)) != EOF)
```

```
    /*  begin while      */
        z[count] = (cost+wl)/1000;     /*  cost  */
        wl=(cost)+wl;
        count++;       /*  increment count  */
    /*  end while               */

fclose(infile);

       strcpy(filename,"graf");
if ((infile = fopen(filename,"r")) == NULL)
            /*  begin if  */
    printf("Sorry, cannot open %s", filename);
    return;
    /*  end if  */

       wl=0;
       counts = 0;
while ((flag = fscanf(infile, "%f%f%f",&center,&budg,&expn)) != EOF)
    /*  begin while      */
        b[counts] = (budg)/1000;       /*  cost  */
        counts++;                      /*  increment count  */
                                    /*  end while   */

fclose(infile);

       settime();   /* Start timing of run. */
       bgnplot(1,'g',"combo.tkf");/* Initialize plot. Graphic mode */

       startplot();

             /* Change to simplex Greek and math */
       font(4,"simplex.fnt",' 310',"duplex.fnt",' 311'
              ,"complex.fnt",' 312',"simgrma.fnt",' 313');
       xlab(&xstring);  /* Turn on string labels */

       cross(0);
       color(0);
       page(4.5,3.4275);
       pgshift(1.0,3.4275);
```

```
        area2d(3.5,2.5);
        nxdiv=16;  /* Desired # of x-axis divisions */
        nydiv=6;   /* Desired # of y-axis divisions */
        npts=8;    /* Number of points in x and y vectors */
        for(i=0;i<=8;i++) a[i]=(float)(i+1);

        color(6);
        grid(2);  /* Put fine dotted grid on plot */
        graf("",0.,1.,9.,"%-1.0f",0.,1.,10.);
        xstring.flag=0;  /* Turn off string label option */
        xlab(&xstring);  /* after axes have been drawn. */
        color(3);
        xname(" 312Cost Function"); /* Label for x-axis */
        heading(" 311BUDGET");      /* Title of plot */
        yname(" 312Millions");      /* Label for y-axis */
        color(2);
        pltfnt(12.5, 8.2, " 312110", .5, 0);

        bar(nxdiv,a,b,npts,2);

/*  full page second line plot  */

         /* Change to simplex Greek and math */
        font(4,"simplex.fnt",' 310',"duplex.fnt",' 311'
                ,"complex.fnt",' 312',"simgrma.fnt",' 313');
        xlab(&mstring);
        cross(0);
        color(0);
        physor(0.0,0.0);   /*RESET DEFAULT ORIGIN*/
        page(9.0,6.855);
        area2d(8.5,6.0);
        box();          /* Draw a box around the plot */
        grid(0);        /* Put fine dotted grid on plot */
        fntchg(' 310'); /* Changes fonts for the axes */
        for(i=1;i<13;i++)

                                /*              budget          */
```

```
                                x[i]=(float)(i+1);

        for(i=1;i<count;i++)

                                /*              budget          */
                                xl[i]=(float)(i+1);

        nxdiv=12;   /* Desired # of x-axis divisions */
        nydiv=5;    /* Desired # of divisions on linear axis */
        npts = 12;
  graf("",0.,1.,12.,"%-1.0f",0.,2.,10.);
        mstring.flag=0;
        xlab(&mstring);
        color(14);
        xname(" 310End of Month"); /* Make labels */
        yname(" 310Millions");
        color(10);
        curve(x,y,npts,0);   /* Draw curve with no symbols */
        chndsh();    /* Use chain-dashed line for second curve */
        color(12);
        curve(xl,z,count,0);/* Plot second curve */
        endplot();          /* Terminate second plot */

        stopplot();         /* Close files and quit */

             /*  end main  */
```

7. TRIPBAR.C

```
/*************************************************************/
/*                  program module TripBar.c                 */
/*                      version 1.0                          */
/*            authors:  Richard N. Woodman                   */
/*                      Michael F Rall                       */
/*                                                           */
/*          Program last modified 20 January 1986           */
/*                                                           */
```

```
/*  This program was produced on an IBM clone using        */
/*  DOS 3.1.  Written with the C programming language,     */
/*  utilizing the GraphicC utility software.               */
/*                                                         */
/*  This is called directly from DOS after the PROJ        */
/*  system has been processed.  This module produces a     */
/*  Triple bar graph. The middle bar represents the budget */
/*  for each cost center, left bar represents the expenses */
/*  and the right bar represents the percentage of the     */
/*  budget expended.                                       */
/*                                                         */
/*  Files used:  GRAF                                      */
/*                                                         */
/*  External Calls:  None                                  */
/*                                                         */
/***********************************************************/


#include "stdio.h"
#include "graphics.h"

int _stack = 60000;

main(argc,argv)
   int argc;
   char *argv[];
                        /*  begin main  */

   FILE *infile, *outfile, *fopen();

                                  /* Declare variables */
          char month[3], filename[30], name[4];
          float cost, x1[14],x2[14],y1[14],x[301], y[301], z[301];
          float a[301], b[14], c[301], d[301];
          int cf, flag, i, count, nxdiv, nydiv, npts;
          long gettime();
          float  budget, budget1, budget2, z1[13], w, w1;
```

```
struct xlab     /* begin Needed for string labels. */
int flag;
char s1[10];
char s2[10];
char s3[10];
char s4[10];
char s5[10];
char s6[10];
char s7[10];
char s8[10];
char s9[10];
char s10[10];
 };                        /* end Needed for string labels. */
#if CIQ /* Some CI86 compilers won't accept the simpler form. */
static struct xlab xstring;
xstring.flag=1;
strcpy(xstring.s1," 310112");
strcpy(xstring.s2," 310113");
strcpy(xstring.s3," 310114");
strcpy(xstring.s4," 310115");
strcpy(xstring.s5," 310116");
strcpy(xstring.s6," 310117");
strcpy(xstring.s7," 310118");
strcpy(xstring.s8," 310119");
strcpy(xstring.s9," 310112");
strcpy(xstring.s10," 310113");
#else
static struct xlab xstring =            /*  begin  */
            1,"","," 310112"," 310113"," 310114"," 310115"
               ," 310116"," 310117" ," 310118"," 310119","" ;
#endif


   strcpy(filename,"graf");
   if ((infile = fopen(filename,"r")) == NULL)
```

```
        /*  begin if  */
        printf("Sorry, cannot open %s", filename);
        return;
        /*  end if  */

           for(i=0;i<=9;i++)

                       a[i]=0;
                       b[i]=0;
                       c[i]=0;
                       d[i]=0;

           count = 0;
   while ((flag = fscanf(infile, "%d%f%f", C.7,&budget,&cost)) != EOF)
                                        /*  begin while      */
            a[count] = cost/1000;          /*  cost                */
            c[count] = budget/1000;        /* cost function budget */
            d[count] = (cost/budget)/100;  /* cost function % of budget */
     count++;                              /*  increment count  */
                                           /*  end while */

    fclose(infile);

           /* These are the strings. */
           settime();     /* Start timing of run. */
           bgnplot(1,'g',"tripbar.tKf");/* Initialize plot. Graphic mode */

/*  begin full page bar plot  */

           startplot(7);

            /* Change to simplex Greek and math */
           font(4,"simplex.fnt",' 310',"duplex.fnt",' 311'
                   ,"complex.fnt",' 312',"simgrma.fnt",' 313');
           xlab(&xstring);   /* Turn on string labels */

           cross(0);
           color(0);
```

```
physor(0.0,0.0);   /*RESET DEFAULT ORIGIN*/
page(9.0,6.855);
area2d(7.5,6.0);
box( );          /* Draw a box around the plot */
nxdiv=24;        /* Desired # of x-axis divisions */
nydiv=10;        /* Desired # of y-axis divisions */
npts=8;          /* Number of points in x and y vectors */
for(i=0;i<9;i++) x1[i]=(float)(i+1);
for(i=0;i<9;i++) x[i]=(float)(i+.75);
for(i=0;i<9;i++) x2[i]=(float)(i+1.25);

color(6);
grid(2);    /* Put fine dotted grid on plot */
graf("",0.,1.,9.,"%-1.0f",0.,1.,10.);
xstring.flag=0; /* Turn off string label option */
xlab(&xstring); /* after axes have been drawn. */
color(3);
xname(" 312Cost Function"); /* Label for x-axis */
heading(" 311BUDGET");      /* Title of plot */
yname(" 312Millions");      /* Label for y-axis */
color(2);
pltfnt(7.5, 10.2, " 310110", .5, 0);

bar(nxdiv,x1,a,npts,2);
bar(nxdiv,x,c,npts,4);
bar(nxdiv,x2,d,npts,5);

endplot( );           /* Terminate first plot */

stopplot( );         /* Close files and quit */


     /*  end main  */
```

LIST OF REFERENCES

1. Keen, Peter G. W. and Michael S. Scott Morton *Decision Support Systems: An Organizational Perspective*, Addison-Wesley Publishing Company, Inc., Menlo Park, CA 1978.

2. Sprague, Ralph H. and Eric D. Carlson *Building Effective Decision Support Systems*, Prentice-Hall, Inc., Englewood Cliffs, NJ 1982.

3. Yourdon, E., *Managing the Structured Techniques: Srategies for Software Development in the 1990's*, Yourdon Press, NY, 1986.

4. De Marco, T., *Structured Analysis and System Specification*, Yourdon Press, NY, 1978.

5. *Management Analysis of the Navy Industrial Fund Program: Shipyard Review Report Draft*, Coopers and Lybrand, August 1985.

6. *NAVSEA Navy Industrial Fund Financial Management Systems and Procedures Manual*, NAVSEAINST 7600.27, Washington, DC, October 1985.

7. *NAVCOMPT Manual*, Washington, DC, 1985.

8. Davis, William *System Analysis and Design: A Structured Approach*, Addison-Wesley Publishing Co., Menlo Park, CA 1983.

9. Ariav, Gad and Michael J. Ginzberg, "DSS Design: A systematic View of Decision Support," *Decision Making: An Interdisciplinary Inquiry*, ed. Gerardo R. Ungson, Kent Publishing Co., NY, 1982.

10. Huber. George P. "Decision Support Systems: Their Present Nature and Future Applications," *Decision Making: An Interdisciplinary Inquiry*, ed. Gerardo R. Ungsun, Kent Publishing Co., NY, 1982.

11. Management Engineering and Information Office *Mare Island Naval Shipyard Information Resouce Management Plan, FY 87* Vol I, May 1986.

12. *TEL-A-GRAF User's Manual*, Version 4.0, ISSCO, Integrated Software Systems Corporation, San Diego, CA 1981.

13. Kroenke, D. M. *Database Processing: Fundamentals, Design, Implementation*, Chicago, IL, SRA, Inc., 1983.

14. *The CPL User's Guide IDR4302*, Prime Computer, Inc., MASS, December 1980.

15. T-378-380 *Prime Computer Training Manual For the Beginner, Mare Island Naval Shipyard*, Code 380.16, 1984.

16. *Oracle User Manual* (*Vol 1*), Oracle Corporation, Belmont, CA, 1983.

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Technical Information Center Cameron Station Alexandria, VA 22304-6145	2
2.	Library, Code 0142 Naval Postgraduate School Monterey, CA 93943-5002	2
3.	CSM, Code 0367 Naval Postgraduate School Monterey, CA 93943-5002	2
4.	Lt. M. F. Rall Information Resource Management (IRM) U. S. Coast Guard MLC (East) Governors Island, NY 10004	2
5.	Capt. R. N. Woodman CMC Code TPI-64 United States Marine Corps, Headquarters Washington, DC 22214	2
6.	Information Resource Manager Management Engineering and Information Office Mare Island Naval Shipyard Vallejo, CA 94592	2

Thesis
R1495 Rall
c.1 DSS development effects
at the Mare Island Naval
Shipyard.